A MARSHALL CAVENDISH **33** COMPUTER COURSE IN WEEKLY PARTS

# INPUT

## LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00 Republic of Ireland £1.25 Malta 85c Australia $2.25 New Zealand $2.95

# INPUT

Vol. 3 No 33



## INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

**PICTURE CREDITS**
Front cover, Projection Audio Visual. Pages 1017, 1018, 1019, 1020, 1021, Paul Chave/Spectrum. Pages 1022, 1024, 1027, Phil Dobson. Pages 1029, 1033, Spectrum. Pages 1031, 1044, 1047, 1048, Peter Reilly. Pages 1034, 1037, 1038, 1041, 1042, Gary Wing. Pages 1045, 1046, Projection Audio Visual.



Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA. England. Typeset by MS Filmsetting Limited, Frome, Somerset. Printed by Cooper Clegg Web Offset Ltd, Gloucester and Howard Hunt Litho, London.




## HOW TO ORDER YOUR BINDERS

**UK and Republic of Ireland:**
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

**Australia:** See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

**New Zealand:** See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

**Malta:** Binders are available from local newsagents.

*There are four binders each holding 13 issues.*

## BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

**UK and Republic of Ireland:**
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

**Australia, New Zealand and Malta:**
Back numbers are available through your local newsagent.

## COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

**HOW TO PAY: Readers in UK and Republic of Ireland:** All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
*Marshall Cavendish Partworks Ltd.*

**QUERIES:** When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

## INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

# YOURS FOR YEARS TO COME

- USING THE PROGRAM
- LOOK AT THE CALENDAR
- LOOK AT THE DIARY
- ADDING ENTRIES
- REVIEWING THE LISTS

**The combined calendar and diary program will help to keep your affairs in order and plan out the coming year. Make an early New Year's Resolution and use it now**

The program lines given here add some more routines to those given last time. So LOAD in the first part and enter the new lines now.

When you first RUN the program you are asked if you have any diary lists already saved. You won't have at this stage, so answer N. The program goes on to display the main menu. The menu for the Spectrum, Acorn, Dragon and Tandy has nine options:

1 Look at monthly calendar
2 Look at year calendar
3 Look at month diary
4 Review/edit finance
5 Review/edit appointments
6 Review/edit celebrations
7 Review/edit holidays
8 Save the lists
9 Leave the program

The program for the Commodore offers the same number of options but they are grouped together slightly differently and only four appear on the main menu. These are:

1 Year calendar
2 See diary
3 Alter diary
4 Leave program

You can look at the monthly or year calendars without entering any data so try these out first.

## LOOK AT THE CALENDAR

Choose option 1 and you can see a calendar for any month of any year within the limits of the program. The limits for the Spectrum, Commodore, Dragon and Tandy are from 1753, when the modern Gregorian calendar started, and the year 29,999. The Acorn is slightly different in that it only works up to 3299, due to the way the date is stored. However, that should be far enough into the future for anyone! The month, by the way, should always be entered as a number (from 1 to 12) the program won't accept words.

If you have a printer attached to your computer you can choose to have a printout of the calendar, otherwise it is just displayed on the screen. The name of the month, and the year are printed at the top and the days and dates are underneath. The week starts with Sunday which is printed in red on some of the computers. The date of Easter Sunday is calculated automatically and is printed out underneath if it falls in the month displayed (either March or April).

There are several things you can do while looking at the monthly calendar, and the range of options are listed at the bottom of the screen (or on the previous page on the Dragon and Tandy). Pressing BREAK, ESCAPE or CLEAR (or M on the Commodore) takes you to the menu. Other keys allow you to step forwards or backwards by one month. The Spectrum uses the Z and X keys, the Commodore uses L and N, the Acorn uses the left and right cursor keys and the Dragon and Tandy use the up and down arrow keys.

The other keys that do something are F, A, C and H which highlight entries relating to Finance, Appointments, Celebrations and Holidays. But they won't show anything until you've made some entries using the next options.

The Commodore has one extra facility—S for swap—that lets you swap back and forth between the monthly calendar and the monthly diary planner.

## WRITING UP THE DIARY

Option 2 on the Commodore and option 3 on the others let you see the monthly diary. But first you should make some entries. On the Commodore choose option 3 followed by the category, on the other machines simply choose the



category directly from the main menu.

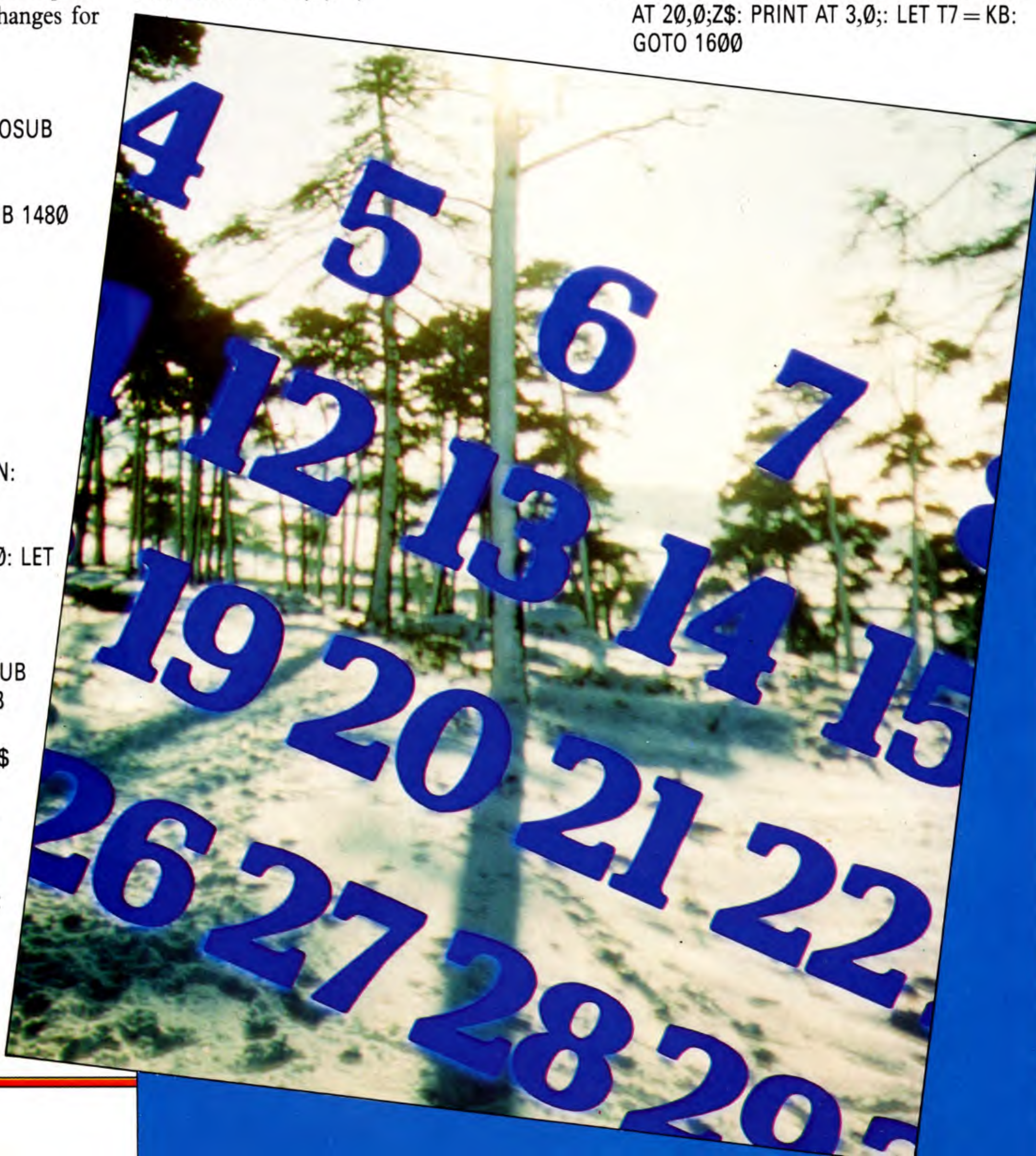
Whichever option you choose, press A to add an entry, D to delete an entry and M to return to the menu.

As described last time, the Finance section offers a choice of Monthly, Quarterly, Annual or Single entries. Make your choice by pressing the initial letter—either M, Q, A or S. You are then prompted to enter a name or sentence of up to 20 letters to describe the entry, followed by the first significant date. For a recurring entry, this would be the first time it occurred—say the first rate payment or your first salary cheque. The program automatically fills in details for the following months and years. Appointments and Holidays are treated as single events but Celebrations are taken as annual so use this option for birthdays and anniversaries.

You can make up to 150 entries in each category on the Spectrum, Acorn and Dragon, and 1ØØ entries on the Commodore.

Next time you can add the final part of the program which lets you SAVE, LOAD and print out the lists. There will also be changes for disk drives and for the Electron.

```
165 GOSUB 1480:IF KB=1 THEN GOSUB
   169Ø
112Ø PAPER Ø: INK 7
113Ø LET K$="123456789": GOSUB 148Ø
   LET C=KB:RETURN
114Ø LET KK=KB
115Ø LET KB=KK: GOSUB 133Ø
116Ø LET B=Q(KK):FOR y=4
   TO 2Ø:PRINT AT y,Ø;Z$:
   NEXT y: PRINT AT 4,Ø
117Ø IF B=Ø THEN GOTO 121Ø
118Ø FOR N=1 TO B
119Ø LET K$=L$(KK,N): LET K2=N:
   GOSUB 137Ø
12ØØ NEXT N
121Ø LET K$="adm": GOSUB 148Ø: LET
   A=KB
122Ø FOR I=1 TO 1ØØ: NEXT I
123Ø IF A<>1 THEN GOTO 128Ø
124Ø LET K2=B: LET T7=KK: GOSUB
   155Ø: LET V$=STR$ T7: GOSUB
   14ØØ
125Ø LET W$=STR$ DA: IF LEN W$
   =1 THEN LET W$="Ø"+W$
126Ø LET V$=V$+W$: LET W$=
   STR$ MO: IF LEN W$=1 THEN
   LET W$="Ø"+W$
127Ø LET V$=V$+W$+STR$ YR:
   LET L$(KK,B+1)=V$+
   B$: LET Q(KK)=Q(KK)+1
128Ø IF A<>2 THEN GOTO
   131Ø
129Ø INPUT "WHICH NUMBER
   TO DELETE (MIN 1)"; NN: IF NN<1
   OR NN>B THEN GOTO 129Ø
13ØØ FOR Z=NN+1 TO B: LET L$(KK,
   Z-1)=L$(KK,Z): NEXT Z: LET Q(KK)=
   Q(KK)-1
131Ø IF A<>3 THEN GOTO 116Ø
132Ø RETURN
133Ø PRINT "Current List"'T$(KB)
134Ø PRINT AT Ø,17; INVERSE 1;"A";
   INVERSE Ø;"DD□"; INVERSE 1;"D";
   INVERSE Ø; "ELETE□"; INVERSE 1;"M";
   INVERSE Ø;"ENU"
135Ø PRINT AT 3,Ø;
136Ø RETURN
137Ø LET F$=L$(KK,K2): LET E$=P$(VAL
   F$( TO 1))+" "+F$(2 TO 3)+
   ":"+F$(4 TO 5)+":"+F$(6 to 9)
138Ø PRINT E$;"□";F$(1Ø TO 3Ø)
139Ø RETURN
14ØØ LET B$="": LET VP=5
141Ø INPUT "To be called ?(Max 22 letters)"'
   LINE B$
142Ø LET VP=VP-1
143Ø PRINT AT VP,Ø;"";
144Ø INPUT "SIGNIFICANT DAY:";DA: IF
   DA<1 OR DA>31 THEN GOTO 143Ø
145Ø GOSUB 251Ø
146Ø LET KB=MO: GOSUB 27Ø: IF DA>KB
   THEN GOTO 143Ø
147Ø LET K$=B$: GOSUB 152Ø: LET
   Y$=K$: RETURN
148Ø LET a$=INKEY$: IF a$="" THEN
   GOTO 148Ø
149Ø FOR n=1 TO LEN K$: IF a$<>K$(n)
   THEN NEXT n
15ØØ IF n>LEN K$ THEN GOTO 148Ø
151Ø LET KB=n:RETURN
152Ø LET PP=(INT (YR/1ØØ)-17)*16+MO
153Ø LET K2=1ØØ: LET QQ=FN M(YR)
154Ø LET K$=CHR$ PP+CHR$ QQ+K$:
   RETURN
155Ø IF T7<>1 THEN GOTO 158Ø
156Ø PRINT AT 2Ø,Ø; INVERSE 1;"M";
   INVERSE Ø;"ONTHLY□"; INVERSE 1;"Q";
   INVERSE Ø; "UARTERLY□"; INVERSE 1;
   "A"; INVERSE Ø;"NNUAL□"; INVERSE 1;
   "S"; INVERSE Ø;"INGLE"
157Ø LET K$="mqas": GOSUB 148Ø: PRINT
   AT 2Ø,Ø;Z$: PRINT AT 3,Ø;: LET T7=KB:
   GOTO 16ØØ
```






```
1580 IF T7=3 THEN RETURN
1590 LET T7=4
1600 RETURN
1610 CLS : PRINT FLASH 1;AT 10,5;
   "PREPARE CASSETTE PLAYER"
1620 IF INKEY$<>"" THEN GOTO 1620
1630 SAVE "data" DATA Q()
1640 FOR N=1 TO 4
1650 IF Q(N)=0 THEN GOTO 1670
1660 FOR M=1 TO Q(N): LET O$(1)=
   L$(N, M):SAVE "data"DATA O$():NEXT M
1670 NEXT N
1680 RETURN
1690 CLS : PRINT AT 10,5; FLASH 1;
   "PREPARE CASSETTE PLAYER"
1700 LOAD "data" DATA Q()
1710 FOR N=1 TO 4
1720 IF Q(N)=0 THEN GOTO 1740
1730 FOR M=1 TO Q(N): LOAD "data"
   DATA O$():LET L$(N,M)=O$(1):NEXT M
1740 NEXT N
1750 RETURN
```

C=

```
1100 DR=1
1110 POKE 53281,3:PRINT"■■
   ■■■"TAB(9)"DIARY EVENTS EDIT
   MODE"
1120 PRINTTAB(9)"■□□□□□□□
   □□□□□□□□□□□□
   □□□■■■"
1130 FOR N=0 TO 3:PRINTTAB(11)
   N+1;"□";ST$(N)"■":NEXT N
1140 PRINTTAB(12)"5□□RETURN TO MENU"
1150 PRINTTAB(13)"■■■■ENTER
   CHOICE ?"
1160 PRINTTAB(13)"■□□□□□
   □□□□□□□□□"
1170 GET A$:IF A$="" GOTO 1170
1180 A=VAL(A$)-1:IF A<0 OR A>4
   GOTO 1170
1190 IF A=4 THEN RETURN
1200 POKE 53281,1:PRINT"■■"
   TAB(20-(LEN(ST$(A))*.5))"■"
   CHR$(CL(A));ST$(A)"■"
1210 MX=VAL(DL$(A,0)):IF MX=0 GOTO
   1290
1220 FOR N=1 TO MX
1230 A$=STR$(N)+"□":IF LEN(A$)<3
   THEN A$=A$+"□"
1240 PRINT A$;
1250 FOR N1=1 TO 4:CD(N1)=ASC
   (MID$(DL$(A,N),N1,1)):NEXT N1
1260 CD(4)=CD(4)+(ASC(MID$
   (DL$(A,N),5,1))*256)
1265 CD(2)=CD(2)-35
1270 PRINTCD$(CD(1));"□";STR$
   (CD(2));"□";STR$(CD(3));"□";
   STR$(CD(4));"□";
1280 PRINT MID$(DL$(A,N),6) NEXT N
1290 PRINTTAB(9)"■(A)DD,
   (D)ELETE,(M)ENU"
1300 GET A$:IF A$="" GOTO 1300
1310 IF A$="M" GOTO 1110
1320 IF A$="A" GOTO 1350
1330 IF A$="D" AND MX>
   0 GOTO 1560
1340 GOTO 1300
1350 PRINT"■■■EVENT
   (16 LETTERS MAX)"
1360 INPUT A$
1370 PRINT"■■■"TAB(12)
   "DIARY ENTRY DATE■■"
1380 INPUT"■■■■■■■
   YEAR■";Y:IF Y<1753 OR
   Y>29999 GOTO 1380
1390 INPUT"■■■■■
   MONTH■";M: IF M<1 OR
   M>12
   GOTO 1390
1400 LY=0
1410 IF M=2 AND Y/4-INT
   (Y/4)=0 THEN LY=1
1420 INPUT"■■■■■■■DAY■";
   D:IF D<1 OR D>VAL(MID$(ML$,
   M*2-1,2))+LY GOTO 1420
1430 IF A=0 GOTO 1480
1440 FR=4:IF A=2 THEN FR=3
1450 MX=MX+1:DL$(A,0)=STR$(MX):
   Y9=INT(Y/256):Y8=Y-(Y9*256)
1460 DL$(A,MX)=CHR$(FR)+CHR$
   (D+35)+CHR$(M)+CHR$(Y8)+
   CHR$(Y9)+LEFT$(A$,16)
1470 GOTO 1200
1480 PRINT"■■■■(M)ONTHLY,
   (Q)UARTERLY,(A)NNUAL,
   (S)INGLE"
1490 FR=0:GET X$:IF X$=""
   GOTO 1490
1500 IF X$="M" THEN FR=1
1510 IF X$="Q" THEN FR=2
1520 IF X$="A" THEN FR=3
1530 IF X$="S" THEN FR=4
1540 IF FR=0 GOTO 1490
1550 GOTO 1450
1560 PRINT"WHICH NUMBER":
   INPUT A$:X=VAL(A$)
1570 IF X<1 OR X>MX GOTO 1560
```

```
1580 FOR N=X+1 TO MX:DL$(A,
   N-1)=DL$(A,N):NEXT N
1590 DL$(A,Ø)=STR$(MX-1): GOTO 1200
1600 IF Z=1 THEN RETURN
1610 IF C>3 THEN CC=WH:
   RETURN
1620 IF M=EM AND CD=ED THEN
   CC=CL(4):RETURN
1630 MX=VAL(DL$(C,Ø))
1640 IF MX=Ø THEN CC=WH:
   RETURN
1650 LX=Ø
1660 LX=LX+1
1670 CD(2)=ASC(MID$(DL$(C,LX),2,1))-35
1680 IF CD<>CD(2) THEN CC=WH:
   GOTO1710
1690 GOSUB1740
1700 IF FL=1 THEN CC=CL(C):
   RETURN
1710 IF LX<MX GOTO1660
1720 CC=WH
1730 RETURN
```

```
125 IF FNget("YN")=1 PROCload
1270 DEF PROClist(type%)
1280 LOCAL n,a%,b%
1290 PROCheader(type%)
1300 REPEAT
1310 CLS
1320 b%=VAL(List$(type%,Ø))
1330 IF b%=Ø GOTO 1370
1340 FOR n=1 TO b%
1350 PROCop(List$(type%,n),n)
1360 NEXT
1370 a%=FNget("ADM")
1380 IF a%=1 List$(type%,b%+1)
   =CHR$(FNtype(type%))+FNadd:
   List$(type%,Ø)=STR$(b%+1)
1390 IF a%=2 AND b%>Ø FOR
   p%=FNnoln(1,b%,"Which number")+1
   TO b%:List$(type%,p%-1)=List$
   (type%,p%):NEXT:List$(type%,Ø)=STR$
   (b%-1)
1400 UNTIL a%=3
1410 ENDPROC
1420 DEF PROCheader(t%)
1430 LOCAL y
1440 FOR y=1 TO 2:PRINTF$+
   Type$(t%):NEXT
1450 PRINTTAB(2,3)CHR$134+
   "Current List"
1460 FOR y=21 TO 24
1470 PRINTTAB(Ø,y)CHR$132+
   CHR$157+CHR$135TAB(35,y)
   CHR$156;
1480 NEXT
1490 PRINTTAB(4,23)"Add Delete Menu";
1500 VDU28,Ø,21,39,4
1510 ENDPROC
1520 DEF PROCop(a$,no%)
1530 LOCALn%,b$,d$
1540 FOR n%=1 TO 4:code%(n%)
   =ASC(MID$(a$,n%,1)):NEXT
1550 b$=MID$(Pay$,code%(1)*4-3,4)
1560 d$=STR$code%(2)+":"+
   STR$(code%(3)MOD16)+":"+
   STR$((code%(3)DIV16+17)*
   100+code%(4))
1570 PRINT;no%;CHR$131;b$;CHR$
   133;d$;TAB(17,VPOS);CHR$135;
   RIGHT$(a$,LENa$-4)
1580 ENDPROC
1590 DEF FNadd
1600 LOCAL b$,vpos%
1610 PRINT"To be called ? (max 2Ø
   letters)":INPUT,b$
1620 vpos%=VPOS+1
1630 REPEAT PRINTTAB(Ø,vpos%);
1640 PRINT"Significant data ?"
1650 Day%=FNnoln(1,31,"□□Day:")
1660 PROCmydate
1670 UNTIL Day%<=FNmonthL
   (Month%)
1680 =FNcode(b$)
1690 DEF FNget(a$)
1700 LOCAL b$,a%
1710 REPEAT b$=GET$:a%=INSTR
   (a$,b$):UNTIL a%
1720 =a%
1730 DEF FNnoln(min,max,b$)
1740 LOCAL y,a$
1750 y=VPOS
1760 REPEAT PRINTTAB(Ø,y)SPC3Ø
1770 PRINTTAB(Ø,y)b$;:INPUTa$
1780 UNTIL VALa$>=min□AND
   VALa$<=max
1790 =VALa$
1800 DEF FNcode(b$)
1810 LOCALp%,q%
1820 p%=(Year%DIV1ØØ-17)*16+
   Month%
1830 q%=Year%MOD1ØØ
1840 =CHR$Day%+CHR$p%+CHR$q%+
   LEFT$(b$,2Ø)
1850 DEF FNtype(t%)
1860 IF t%=Ø PRINT"Monthly,
   Quarterly,Annual,Single":=
   FNget("MQAS")
1870 IF t%=2 =3
1880 =4
1890 DEF PROCsave
1900 LOCALn%,p%
1910 X=OPENOUT("Diary")
1920 FOR n%=Ø TO 3
1930 FOR p%=Ø TO VAL(List$(n%,Ø))
1940 PRINT#X,List$(n%,p%)
1950 NEXT,
1960 CLOSE#X
1970 ENDPROC
1980 DEF PROCload
1990 LOCAL X,n%,p%,m%
2000 X=OPENIN("Diary")
2010 FOR n%=Ø TO 3
2020 INPUT#X,List$(n%,Ø)
2030 m%=VAL(List$(n%,Ø))
2040 IF m%=Ø GOTO2080
2050 FOR p%=1 TO m%
2060 INPUT#X,List$(n%,p%)
2070 NEXT
2080 NEXT
2090 CLOSE#X
2100 ENDPROC
2110 DEF PROCannual
2120 LOCALm%,a
2130 Year%=FNnoln(1752,3299,
   "□Year:"):PROCeaster
2140 PROCprinter:CLS
2150 PRINTF$;"YEAR□";Year%;
   TAB(35)CHR$156
2160 VDUP%:PRINTF$;"YEAR□";
   Year%;TAB(35)CHR$156
2170 PRINT:PROCprintdays(Ø):PRINT
2180 PROCspacebar
2190 FOR Month%=1 TO 12
2200 PRINTCHR$129;MID$(Month
   Name$,Month%*9-8,9)
2210 PROCprintmonth(5,Ø)
2220 IF P%=Ø a=GET
2230 NEXT
2240 VDU26,3
2250 ENDPROC
```

```
115 KB$="YN":GOSUB 159Ø:IF KB=1
   GOSUB 187Ø
1180 'LIST & UPDATE
1190 N=Ø:A=Ø:B=Ø:KK=KB
1200 KB=KK:GOSUB 133Ø
1210 REM
1220 FOR VU=1 TO 14:PRINT@VU*32:
   NEXT:PRINT@32,"";
1230 B=VAL(LI$(KK,Ø))
1240 IF B=Ø THEN 128Ø
1250 FOR N=1 TO B
1260 KB$=LI$(KK,N):K2=N:GOSUB 141Ø
1270 NEXT
1280 KB$="ADM":GOSUB 159Ø:A=KB
1290 IF A=1 THEN T7=KK:GOSUB 168Ø:
   GOSUB 149Ø:LI$(KK,B+1)=CHR$
   (T7)+T8$:LI$(KK,Ø)=MID$
   (STR$(B+1),2)
1300 IF A=2 THEN INPUT "WHICH
   NUMBER";NN:IF NN<1 OR NN>B THEN
   13ØØ ELSE FOR PP=NN+1 TO
   B:LI$(KK,PP-1)=LI$(KK,PP):
   NEXT:LI$(KK,Ø)=STR$(B-1)
1310 IF A<>3 THEN 121Ø
1320 RETURN
1330 'SET UP HEADER
1340 PRINTTY$(KB),"CURRENT LIST"
1350 FOR Y=2 TO 14
1360 PRINT@Y*32
```

```
1370 NEXT
1380 PRINT@480,"ADD DELETE MENU";
1390 PRINT@32,"";
1400 RETURN
1410 'OP
1420 N2=Ø:BB$="":DD$="":K3=K2
1430 FOR N2=1 TO 4:IF MID$(KB$,
   N2,1)="" THEN CO(N2)=Ø ELSE
   CO(N2)=ASC(MID$(KB$,N2,1))
1440 NEXT
1450 BB$=MID$(PA$,CO(1)*4-3,4)
1460 K2=16:DD$=MID$(STR$(CO(2)),
   2)+":"+MID$(STR$(FNM(CO(3))),2)+
   ":"+MID$(STR$((FIX(CO(3)/16)+17)*
   1ØØ+CO(4)),2)
1470 PRINTMID$(STR$(K3),2);"□";
   BB$;"□";DD$;"□";RIGHT$
   (KB$,LEN(KB$)-4)
1480 RETURN
1490 'ADD AN ENTRY
1500 B3$="":VP=Ø
1510 PRINT"TO BE CALLED ? (MAX 22
   LETTERS)":LINE INPUT B3$
1520 VP=INT((PEEK(136)*256+
   PEEK(137)-1Ø24)/32)
1530 PRINT@VP*32,"";
1540 PRINT"SIGNIFICANT DATE ?"
1550 INPUT "□□DAY:";DA:IF DA<1 OR
   DA>31 THEN 153Ø
1560 GOSUB 275Ø
1570 KB=MO:GOSUB 23Ø:IF DA>KB THEN
   153Ø
1580 KB$=B3$:GOSUB 163Ø:T8$=KB$:
   RETURN
1590 'CHECK KBD FOR CHARACTER IN KB$
   AND RETURN POSITION IN KB
1600 B$=INKEY$:IF B$="" THEN 16ØØ
1610 KB=INSTR(1,KB$,B$):IF KB=Ø THEN
   16ØØ
1620 RETURN
1630 'CODE INFO
1640 PP=Ø:QQ=Ø
1650 PP=(FIX(YR/1ØØ)-17)*16+MO
1660 K2=1ØØ:QQ=FNM(YR)
1670 KB$=CHR$(DA)+CHR$(PP)+CHR$
   (QQ)+LEFT$(KB$,22):RETURN
1680 'GET TYPE
1690 IF T7=Ø THEN PRINT"MONTHLY,
   QUARTERLY,ANNUAL,SINGLE":KB$=
   "MQAS":GOSUB 159Ø:T7=
   KB:GOTO 172Ø
1700 IF T7=2 THEN T7=3:GOTO 172Ø
1710 T7=4
1720 RETURN
1730 'SAVE ARRAY
1740 N=Ø:P=Ø
1750 OPEN "O",#-1,"DIARY"
1760 FOR N=Ø TO 3
1770 M=VAL(LI$(N,Ø))
1780 PRINT#-1,LI$(N,Ø)
1790 IF M=Ø THEN 184Ø
```




```
1800 FOR P=1 TO M
1810 FOR J=1 TO 4:PRINT#-1,STR$
   (ASC(MID$(LI$(N,P),J,1))):NEXTJ
1820 PRINT#-1,MID$(LI$(N,P),5)
1830 NEXTP
1840 NEXTN
1850 CLOSE #-1
1860 RETURN
1870 'LOAD ARRAY
1880 N=Ø:P=Ø:M=Ø
1890 OPEN "I",#-1,"DIARY"
1900 FOR N=Ø TO 3
1910 LINE INPUT #-1,LI$(N,Ø)
1920 M=VAL(LI$(N,Ø))
1930 IF M=Ø THEN 198Ø
1940 FOR P=1 TO M
1950 FOR J=1 TO 4:INPUT#-1,
   NN$:LI$(N,P)=LI$(N,P)+
   CHR$(VAL(NN$)):NEXTJ
1960 LINE INPUT #-1,NN$:LI$(N,P)=
   LI$(N,P)+NN$
1970 NEXT
1980 NEXT
1990 CLOSE#-1
2000 RETURN
2010 'YEARLY CAL
2020 M4=Ø:A4=Ø
2030 INPUT "YEAR:";YR:IF YR<1753 OR
   YR>29999 THEN 2Ø3Ø ELSE GOSUB
   65Ø
2040 GOSUB 272Ø:CLS
2050 PRINT"YEAR□";YR
2060 IF P=2 THEN PRINT#-2,
   "YEAR□";YR
2070 PRINT#-P:KB=Ø:GOSUB 215Ø:
   PRINT#-P
2080 GOSUB 266Ø
2090 FOR MO=1 TO 12
2100 PRINT#-P,MID$(MN$,MO*9
   -8,9)
2110 T2=5:S2=Ø:GOSUB 224Ø
2120 IF P=Ø AND INKEY$=""THEN
   212Ø
2130 NEXT
2140 RETURN
2150 'PRINTDAYS -KB
2160 X2=Ø:C2=Ø:D2=Ø
2170 IF KB=Ø THEN X2=7
2180 IF P=2 THEN KB=KB+1
2190 PRINT#-P,STRING$(X2,32);
2200 FOR D2=Ø TO 6
2210 PRINT#-P,STRING$(KB,"□")+
   MID$(DN$,D2*3+1,3);
2220 NEXT
2230 RETURN
```

# 'FLICKER-BOOK' ANIMATION

**Using the simple techniques of paged graphics will give a real insight into the world of computer animation. And it makes an interesting display for your micro**

All types of animation rely on a phenomenon of perception known as persistence of vision. In effect, this means that an image which we see is 'held' in memory for an appreciable instant, even after the view is changed to something else. If a sequence of still images is shown rapidly, the brain cannot keep up with changes of picture occurring more than twelve or so times a second. As a result, it ceases to see separate images—the pictures blur into one another, and the brain is fooled into seeing movement.

The process is very simply demonstrated by the 'flicker-book', in which drawings on each page of a book can be animated as the pages are flipped over at speed. A more sophisticated demonstration of the same thing is the process of stop-frame animation, which dates back to the early years of this century.

This type of animation consists of drawing a shape on a piece of clear plastic, known as a *cell*, and then photographing two or so frames of film using a conventional cine camera mounted on an overhead rostrum. The cell is then replaced with one showing a slightly altered version of the shape, and the whole process is repeated. As you can imagine, this type of animation requires an incredible number of pictures to be painstakingly drawn by hand, as about twenty-five frames are needed for every second of the finished film.

## COMPUTER GRAPHICS

So why not use computers to speed up the process? Even the relatively humble home micro can produce good pictures, while purpose-built main frames are capable of staggeringly sophisticated images.

The problem is that to produce a display of the quality taken for granted by cinema audiences nowadays requires a fantastically extensive assortment of hardware. One science-fiction film—*The Last Starfighter*—relies on computer equipment to achieve 27 minutes of breathtaking images. But to do this needed a $12 million Cray X-MP interfaced to two $1 million mainframes. This massive expenditure was considered worthwhile as it allowed pictures to be constructed which would be difficult in the real world.

This technology is all very well for people who have access to motion-picture or video recording equipment, powerful computers and plenty of time. But most people who possess none of these things still find computer generated graphics of enormous use. Animated images are an important part of all sorts of Computer-Aided Design (CAD) packages, for example, and every good arcade game relies heavily on smoothly-animated, attractive screen displays.

The sophistication that can be achieved in this is limited by the capabilities of the computer you use. The Cray operates at 1ØØ megaflops (1ØØ million floating point operations per second). But as the images have to be swapped many times a second for realistic animation, even the Cray could not generate film-quality images quickly enough for real-time animation. Instead, the images that it generated were filmed separately in a conventional stop-frame process.

Lower levels of detail in the pictures permit frames to be generated more quickly. Indeed, there exists a flight simulator in which reasonably accurate representations of an aeroplane in flight are generated 5Ø times per second, giving realistic animated effects.

The reason for the difficulty of high-speed computer animation is the amount of information required by a picture. The more detailed the image, the more memory is needed to store it. Increase the number of colours available and the amount of RAM required to store the colour information is increased, too.

The more memory used to store the picture, the more work the CPU has to do to update it. The reason why most commercial computer-generated films use stop-frame animation is simply that the processor is not sufficiently powerful to update large areas of memory quickly. If updating graphic displays is a slow process, even to people with powerful computers, how can a home programmer produce animation using a micro? One solution is the use of paged graphics.

## WHAT ARE PAGED GRAPHICS?

Every home computer has an area of memory associated with the screen. It can either be *memory mapped*, which means that for each screen location there is a corresponding memory location, or it may be organized using a display file.

With paged graphics, instead of building the next picture up directly in the screen memory, an area is reserved somewhere else for this purpose. Once the new picture has been completed, it is copied to the RAM normally associated with the screen. These extra screen

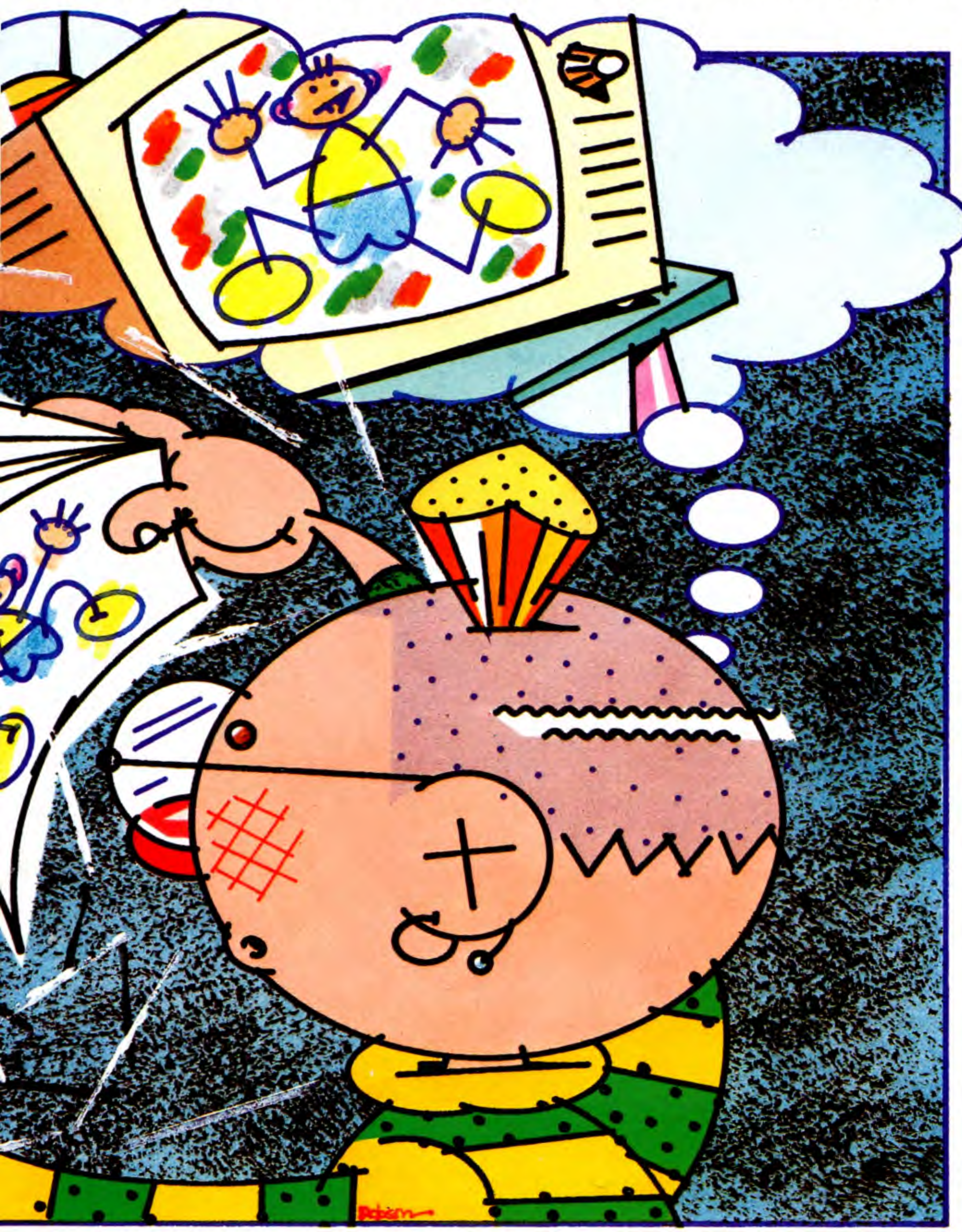


- STOP-ACTION ANIMATION
- COMPUTER GRAPHICS
- PAGED GRAPHICS EXPLAINED
- A MOVING CUBE
- CREATING GRAPHICS
- FURTHER EXPERIMENTS
- REAL TIME ANIMATION
- PERSISTENCE OF VISION
- COMMERCIAL FILM ANIMATION

areas are called pages, which is why the technique is known as paged graphics.

But why do this? Writing to the other page certainly doesn't save any time. In fact, updating the display will take slightly longer because of the time taken to copy the information into the screen memory. The advantage lies in the fact that while updating the 'hidden' page, there is no change on the current screen display.

Obviously, if you only want to draw one picture, paged graphics may seem of little use. However, if you want to write a program in which a page of text is followed by a picture, say, think how convenient it would be to be able to start creating the first graphical display somewhere else in memory while the user is occupied reading the page full of instructions. When the display is needed, time is saved as it is already present in another area of RAM. If convenient you can actually perform all the time consuming calculations yourself, and just use the computer to display the pictures as quickly as necessary.



But the real advantage of paged graphics is if you want to display more than one image in rapid succession. BASIC graphics commands usually only write to the screen memory. This means that pictures are built up on the screen and then saved afterwards. While using paged graphics will not make the physical construction of a screen picture any quicker, once the calculations have been performed, pictures can be recalled from memory to the screen very quickly. Paged graphics thus retain the simplicity of BASIC, combined with much greater display speeds. And if you set up several images in different areas of memory, they can be called up in sequence rapidly to permit relatively complex displays.

## CUBE ALGORITHM

Suppose that you want to set up a simple animated sequence involving the rotation of a cube. You have decided that four pictures will be sufficient to represent one rotation of the cube, and you want the cube to rotate five times. A typical program structure might look like this:

```
for count=1 to 5 do
begin
clear the screen
construct picture number 1
clear the screen
construct picture number 2
clear the screen
construct picture number 3
clear the screen
construct picture number 4
end
```

The idea is simple enough: the screen is cleared, each of the four pictures is drawn in sequence and the process is repeated until the




The Spectrum display: a trampolinist

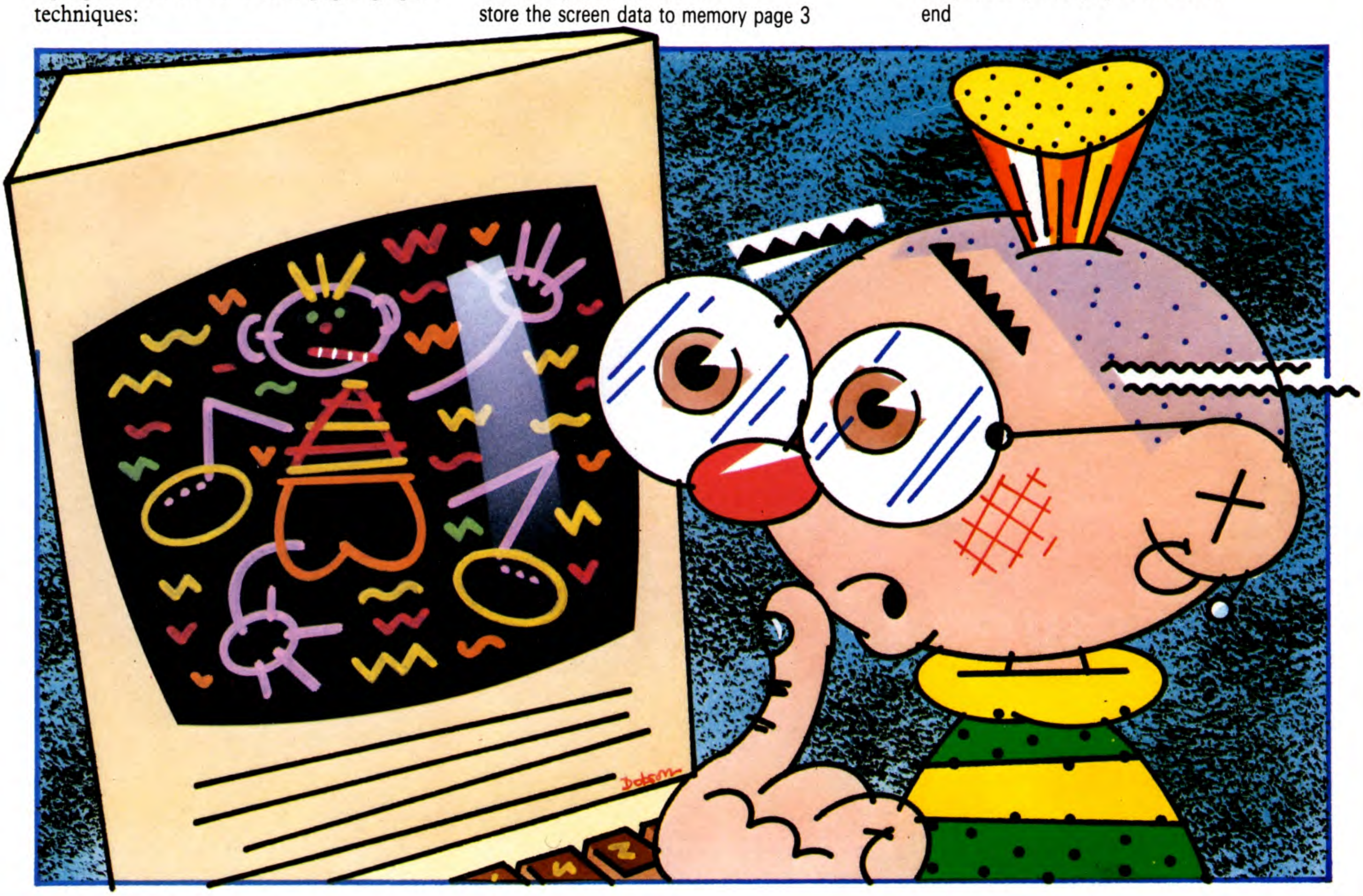


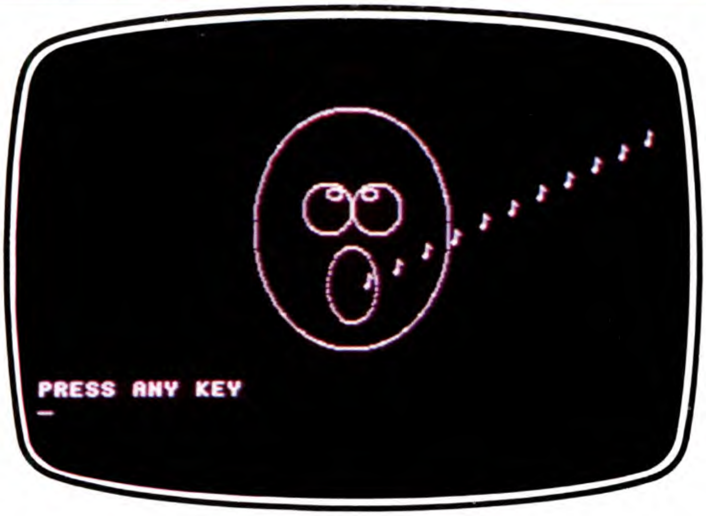

The unusual graphics screen on the Commodore

cube has rotated five times. The disadvantage of this method is that it will draw each picture five times. The calculations each take some time, so poor animation results—with an appreciable 'jump' between each image.

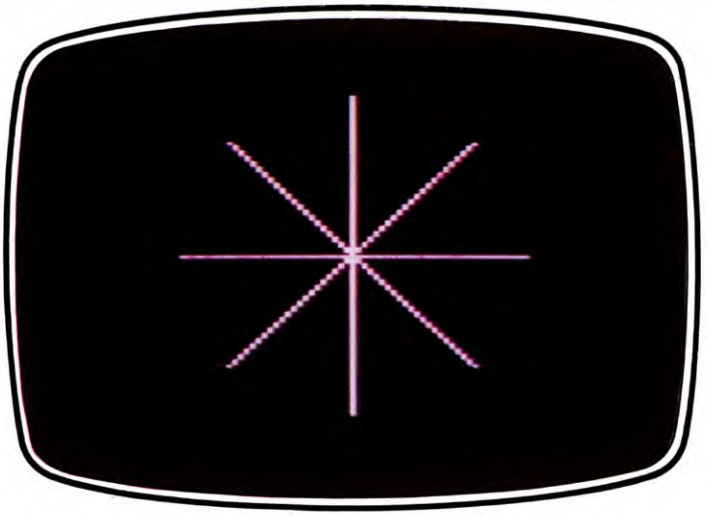
Now look at the alternative algorithm below, which shows the general procedure for a program structured around paged graphics techniques:

```
clear the screen
construct picture number 1
store the screen data to memory page 1
clear the screen
construct picture number 2
store the screen data to memory page 2
clear the screen
construct picture number 3
store the screen data to memory page 3
clear the screen
construct picture number 4
store the screen data to memory page 4
for count = 1 to 5 do
copy memory from page 1 to screen
copy memory from page 2 to screen
copy memory from page 3 to screen
copy memory from page 4 to screen
end
```






**Paged graphics: let's hear it for the Acorn**



**A rotating asterisk makes the Dragon display**

The program is longer and you still have to go through the time-consuming process of calculations and drawing the four pictures, but animation does not start until this has been done. Once stored in memory, the pictures can be recalled very quickly.

Although the drawing is still done in BASIC the actual piece of program to move a particular picture in the memory will be a short machine-code routine. This transfers an entire screen of information faster than the eye can see—the essence of animation.

## GRAPHIC DEMONSTRATION

The following programs demonstrate a simple application of paged graphics on each computer. As the programs contain machine code to get the required speed of exchanging the images, SAVE them before RUNning, in case of mishaps.

In each case you are prompted to press a key after each image has been drawn, and a key has to be pressed to start the images alternating.

You can use these programs as the basis for your own experiments, by changing the images that they draw. But a later article explains the techniques involved in more detail, and how to push the sophistication of the animation to the limits of your computer.

This program—suitable for 48K Spectrums only—will give you a simple animation of a man bouncing on a trampoline. Although the majority of the program is in BASIC, there is some machine code that gives the speed required to call the pages from memory.

```
1Ø BORDER Ø: PAPER Ø: INK 7: CLS
2Ø CLEAR 5323Ø
3Ø GOSUB 22Ø
4Ø LET srce=64: LET dest=2Ø8
5Ø CLS
6Ø CIRCLE 128,168,7: PLOT 128,161: DRAW
   Ø,−15: DRAW −1Ø,−1Ø: PLOT 128,146:
   DRAW 1Ø,−1Ø: PLOT 118,161: DRAW
   11,−5: DRAW 1Ø,5
7Ø PLOT 1Ø8,1Ø6: DRAW 4Ø,Ø: PLOT
   113,1Ø6: DRAW −8,−8: PLOT 145,1Ø6:
   DRAW 8,−8
8Ø GOSUB 27Ø: LET dest=dest+16
9Ø PRINT AT 21,Ø;"any key when ready":
   PAUSE Ø
1ØØ CLS : CIRCLE 128,141,7: PLOT 128,134:
   DRAW Ø,−15: DRAW −5,−16: PLOT
   128,12Ø: DRAW 5,−17: PLOT 118,125:
   DRAW 1Ø,5: DRAW 11,−5
11Ø PLOT 1Ø8,1Ø6: DRAW 15,−4: DRAW
   1Ø,Ø: DRAW 15,4: PLOT 113,1Ø5: DRAW
   −8,−8: PLOT 144,1Ø5: DRAW 8,−8
12Ø PRINT AT 6,4;"!!BOING!!"
13Ø GOSUB 27Ø
14Ø PRINT AT 21,Ø;"any key when ready":
   PAUSE Ø
15Ø LET srce=2Ø8: LET dest=64
16Ø REM PRINT AT 17,Ø;"press any key to
   RESTORE ": PAUSE Ø
17Ø FOR n=Ø TO 1
18Ø CLS
19Ø GOSUB 27Ø: LET srce=srce+16
2ØØ NEXT n
21Ø GOTO 15Ø
22Ø DATA 1,Ø,16,17,Ø,Ø,33,Ø,Ø,237,176,2Ø1
23Ø FOR i=53231 TO 53231+11
24Ø READ byte: POKE i,byte
25Ø NEXT i
26Ø RETURN
27Ø POKE 53236,dest
28Ø POKE 53239,srce
29Ø RANDOMIZE USR 53231
3ØØ RETURN
```

Line 1Ø sets the colours of the screen, border and the display: black, black and white respectively. Line 2Ø reserves an area of memory for the machine code that you will be using and then Line 3Ø sends the program off to a subroutine at Lines 22Ø to 26Ø that sets up the machine code. This subroutine reads off the DATA in Line 22Ø and POKEs it into the memory area reserved by Line 2Ø before returning to Line 4Ø. Line 4Ø defines two variables: srce and dest. srce is the high byte of the address where the DATA is to be taken from and dest is the high byte of the temporary address store. These tell the computer where to read the screen image from and where to put it in memory.

With this out of the way the first of the two images for the two passes is drawn—a man up in the air poised over a trampoline—and this is taken care of in Lines 6Ø and 7Ø. Line 8Ø first sends the program off to Line 27Ø where there is a routine that puts the dest and srce numbers into the machine-code program and then calls the machine code to copy the portion of the screen in which the image of the trampolinist is seen.

The next step is to create the image for the second page that is to be stored. The two lines that achieve this are 1ØØ and 11Ø, while Line 12Ø PRINTs a non-audible sound effect! This second image is stored by Line 13Ø which sends the program to the subroutine at 27Ø and the Line 15Ø swops over the values of srce and dest which has the effect of downloading the image from RAM onto the screen. Lines 15Ø to 21Ø set up a loop that will alternate the two images that have been created. This will RUN until you press the BREAK key.

You can use this program to set up your own two-frame animation by changing the graphics commands in lines 6Ø and 7Ø, and

lines 1ØØ and 11Ø to make new images. But under the right circumstances it is possible to get as many as eight pages to run in sequence. The program to do this becomes rather more complicated, and is covered in detail in a later article.

Use of the Commodore's excellent resolution graphics—accessible by using either a Simon's Basic Cartridge or INPUT's high-resolution graphics facility starting on page 748—gives an unusual and interesting alternating pair of images.

The two images are similar, and so they are not drawn individually. Instead, thay are constructed using a FOR... NEXT loop which sends the program through the drawing routine twice, making small changes to the instructions the second time around.

When using a Simon's Basic Cartridge the program below is correct. If using INPUT's high resolution graphics facility, as has been published so far on pages 748 to 751 and 872 to 877, delete Line 65, change the **224** in bold in Line 24Ø to 32 and the **255** in bold in Line 25Ø to 63. When the remainder of INPUT's high-res facility is published, Line 65 can be left in but the changes to Lines 24Ø and 25Ø must still be made. Also, you must preface all the graphics commands with @, as explained on page 748.

```
20 POKE 51,255:POKE 52,29:POKE
   55,255:POKE 56,29:CLR
30 GOSUB 220
40 D=64
50 FOR N=0 TO 1
60 HIRES 0,1:MULTI 2,4,5:COLOUR6,6
65 CIRCLE 80,100,30+10*N,30+10*N,
   1:PAINT 80,100,2
70 FOR X=0 TO 159:PLOT
   RND(1)*160,RND(1)*200,RND(1)*4
80 PLOT X,100+30*SIN((N+X/50)*π/4),3
85 LINE X, 50+N*50, 0, 100- N*50, RND (1)
   *4+1
90 NEXT X
100 GOSUB 430:IF N=0 THEN D=96
110 TEXT 30,1,"PRESS ANY KEY",3,1,8:POKE
   198,0:WAIT 198,1
120 NEXT N
150 D=64:FOR N=0 TO 1
170 GOSUB 440:IF N=0 THEN D=96
190 NEXT N
200 GOTO 150
220 FORZ=7680 TO 7738:READ X:POKE
   Z,X:NEXT Z:RETURN
230 DATA 169,0,141,14,220,169,53,133,1
240 DATA 169,0,133,251,133,253,169,224,
   133,252,169,64,133,254,160,0
250 DATA 177,251,145,253,192,63,208,16,
   165,252,201,255,208,10
260 DATA 162,1,142,14,220,162,55,134,1,96,
   200
270 DATA 208,229,230,252,230,254,
   76,25,30
430 POKE 7700,D:POKE 7706,251:POKE
   7708,253:SYS 7680:RETURN
440 POKE 7700,D:POKE 7706,253:POKE
   7708,251:SYS 7680:RETURN
```

Line 2Ø of the program clears some space in the memory for the machine code and the screens that will need to be stored. Line 3Ø then sends the program to the subroutine at Lines 22Ø to 27Ø, where the DATA is POKEd into memory. Line 4Ø sets the position in the memory where the first of the screens is to stored and 5Ø is a FOR...NEXT loop for the different screens. The graphics mode and screen colour (blue) are set in Line 6Ø.

The first and second of the designs is drawn by Lines 65 to 85. In Line 1ØØ the program is first of all sent to Line 43Ø which saves the screen before changing the screen to its location. The two screens are swopped over by a routine contained in Lines 15Ø and 2ØØ.

The end result of the Acorn program is a singer giving his all and sending notes wafting into the air. This is achieved using two pages of graphics in which only the man's mouth, eyes and the musical notes move. The programming to produce these graphics on screen, however, takes up a number of lines as there are several shapes to be drawn (using a simple ellipse routine).

```
10 VAR=0
20 MODE4
30 VDU 23,224,16,24,20,16,16,48,112,32
40 HIMEM=&3000:DIM MC% 100
50 PROCCODE
60 F%=&58:T%=F%-&14
70 FOR N=0 TO 1
80 CLS
90 PROCC(7.8,.018,200,640,240,752):PROCC
   (7.8,.08,50,590,50,790):PROCC(7.8,.08,
   50,690,50,790)
100 IFN=1 THEN 160
110 PROCC(7.8,.08,50,640,75,640)
120 PROCC(7.8,.08,15,605,10,820)
130 PROCC(7.8,.08,15,675,10,820)
140 PROCNOTES(0)
150 GOTO 190
160 PROCC(7.8,.08,75,640,50,640)
170 PROCC(7.8,.08,15,605,10,775):PROCC
   (7.8,.08,15,675,10,775)
180 PROCNOTES(90)
190 PROCMOVE(F%,T%):T%=T%-&14
200 PRINTTAB(0,18);"PRESS ANY KEY":
   A$=GET$
210 NEXT
220 PRINTAB(0,18)SPC(15)
230 T%=&58:F%=T%-&14
240 FOR N=0 TO 1
250 PROCMOVE(F%,T%):F%=F%-&14
260 D=INKEY(25)
270 NEXT N
280 GOTO 230
290 DEF PROCCODE
300 REM BLOCK MOVE CODE
310 FOR OPT%=0 TO 2 STEP 2
320 P%=MC%
330 [
340 OPT OPT%
350 .MVBLK□LDA #19
360 JSR &FFF4
370 LDX#&14
380 LDY#0
390 .NXT□LDA(&70) ,Y
400 STA (&72) ,Y
410 DEY
420 BNE NXT
430 INC &71
440 INC &73
450 DEX
460 BNE NXT
470 RTS
480 ]
490 NEXT OPT%
500 ENDPROC
510 DEF PROCMOVE(F%,T%)
520 REM CALL BLOCK MOVE CODE
530 ?&70=0:?&71=F%
540 ?&72=0:?&73=T%
550 CALL MVBLK
560 ENDPROC
570 DEFPROCC(A,B,C,D,E,F)
580 FOR X=1 TO A STEP B:PLOT
   69,SIN(X)*C+D,COS(X)*E+F
590 NEXT
600 ENDPROC
610 DEF PROCNOTES(F)
620 VDU 5
630 FOR T=1 TO 11
640 MOVE600+T*60+F,630+T*30+F/2
650 VDU 224
660 NEXT
670 VDU 4
680 ENDPROC
```

Line 2Ø sets the MODE. A UDG for the notes is established by Line 3Ø. The next line, Line 4Ø allocates the area of memory into which the two images are to be moved, above that of the BASIC program and, then puts aside some memory for the machine code. The PROCCODE in Line 5Ø calls the routine to set up the machine code. This machine code moves the blocks of memory around.

The next thing to do is to define the addresses of the area from which the block is to be taken and to which it is to be directed. Line 6Ø gives the high byte address, &58, of where the memory is to be taken F(%)rom is assigned. This is called F%. The position where it is to go T(%)o, T%, is also given a high byte address F% − &14. The program now enters a short loop at Line 7Ø and after the screen is cleared in Line 8Ø, the fixed parts of the singers face are drawn by Line 9Ø and the subroutine at Lines 57Ø–6ØØ. As N is only equal to Ø on this first time round the loop, the program continues past Line 1ØØ to Lines 11Ø to 13Ø where the first of the two sets of the singers' moving features are drawn. Line 14Ø then calls Line 61Ø to draw the first set of notes. The program returns to Line 19Ø which first puts this completed first scene into the page assigned in RAM, then moves the pointer down to the second position in RAM where the next face is to be stored.

Line 2ØØ gives the PRESS ANY KEY message and waits for the key press—when this is detected, the program is sent to Line 8Ø by Line 21Ø. The program is now going round this loop for the second time and after the screen is cleared in Line 8Ø, the fixed face features are drawn in Line 9Ø again. Now, as N is equal to one, Line 1ØØ sends the program to Line 16Ø. This and the next line draw the second of the two sets of moving features. Then in Line 18Ø, PROCNOTES (9Ø) calls the PROCedure to draw the notes again, but this time in a different position, thanks to the (9Ø) fed into the F variable of the PROCedure.

Line 19Ø now stores this second face in the previously reassigned position. Once more, 2ØØ waits for a key press, but this time as the loop is finished the program proceeds to Line 22Ø, where PRESS ANY KEY is deleted from the screen, and then onto a loop starting at Line 23Ø and ending at Line 28Ø. The PROCMOVE is used to call the first image to the screen, and the program pauses for $\frac{1}{4}$ of a second (Line 26Ø) puts it back into RAM and then calls the second image onto the screen and again shows it with a delay of $\frac{1}{4}$ of a second. This will keep on going until you press the ESCAPE key.

The program for the Dragon and Tandy is slightly different from that used on the other machines for two reasons. For a start, it uses three, not two, pages and also, because the permanent software for these machines has already got the facility for eight pages of paged graphics—accessible through the PCOPY command—without machine code. The command has the simple form: PCOPY number of first page TO number of second page—PCOPY 1 TO 8, for example. In fact the alternation of the pages is so fast that a delay has to be built into the program to stop the images from being displayed too quickly.

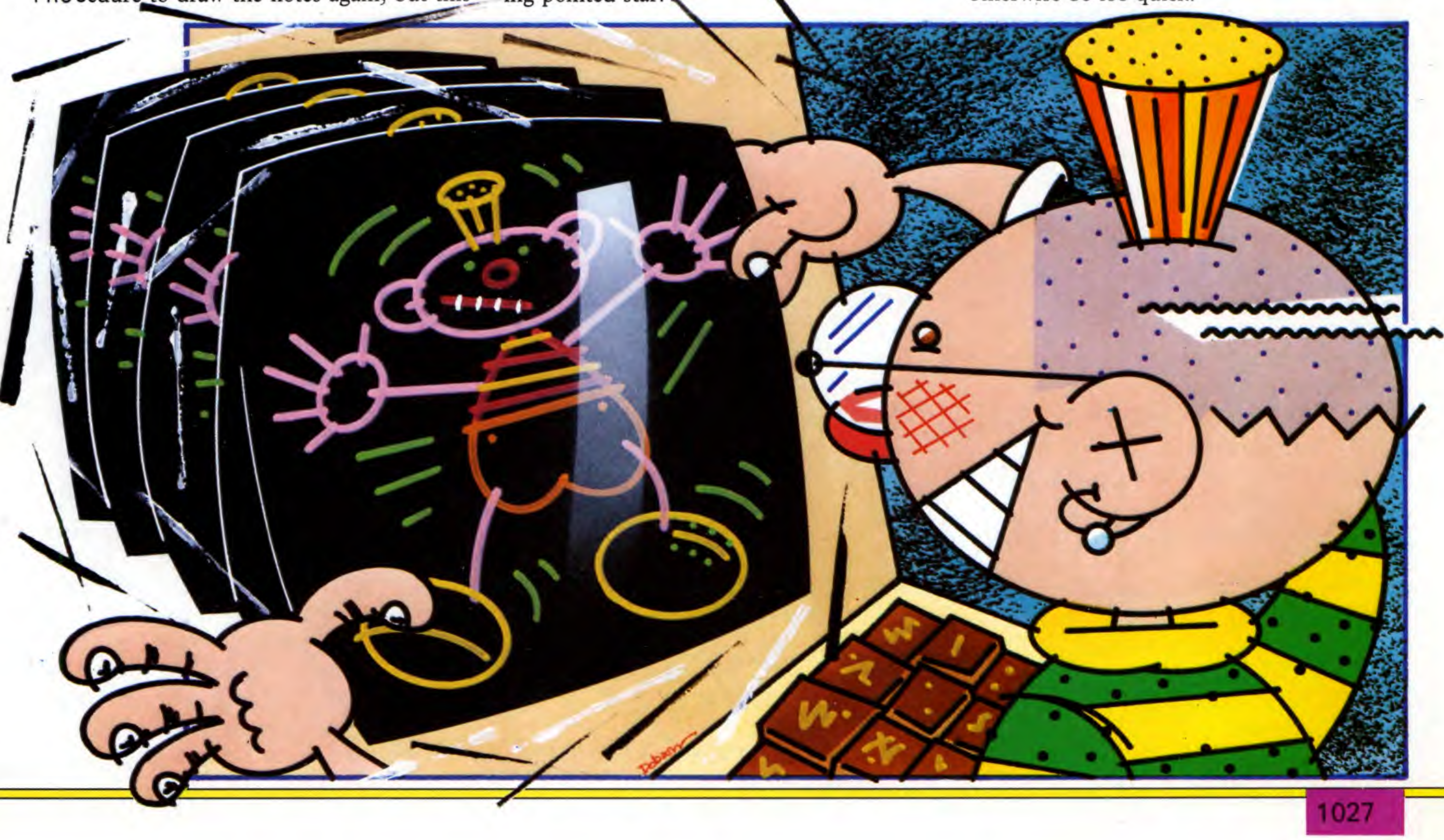
This program gives a fairly smoothly rotating pointed star.

```
1Ø PCLEAR8:PMODE2,1
2Ø SCREEN1,1:CLS
3Ø C=ATN(1)/45
5Ø FORN=ØTO2
6Ø PCLS
7Ø FORK=ØTO36Ø STEP45
8ØLINE(127,95)-(127+59*SIN
   (C*(K+N*15)),95-59*COS
   (C*(K+N*15))),PSET
9Ø NEXT
1ØØ PCOPY1TO3+N*2:PCOPY2TO4+N*2
11Ø NEXT
14Ø FORN=3TO7 STEP2
15Ø PCOPYN TO1:PCOPYN+1TO2
16Ø FORG=1TO3Ø:NEXTG,N
17Ø GOTO14Ø
```

Line 1Ø allocates the eight pages to graphics and selects PMODE 2 at page 1 to give black and white with medium resolution. In this mode, one screenful uses two of the internal graphics pages. In Line 2Ø the high resolution on graphics are turned on.

The images that are to make up the three pages are set up by Lines 3Ø to 11Ø while Line 1ØØ PCOPYs each of these different images onto the internal graphics pages. Each of the screen displays takes up two of the internal pages and the first two pages of the internal pages are used to design the graphics.

Lines 14Ø to 16Ø copy the stored pages into the display in sequence. There is a delay built into Line 16Ø because the alternation would otherwise be too quick.



# ARMING THE BANDIT

**Spin the reels and nudge yourself in the direction of Las Vegas with *INPUT*'s Fruit Machine. Playing this machine won't put a squeeze on your pocket**

Old fashioned mechanical fruit machines are being replaced by modern electronic ones in many arcades, with a television screen being used to display the reels. This two-part article in Games Programming will show you how to imitate one of these machines using your micro.

The game has all the features you would expect in a real fruit machine—hold, gamble, nudge, and so on—and there are animated graphics to simulate mechanical reels.

Having a program like this to play with saves you losing huge amounts of money to the grasping one-armed bandits—although on the other hand, you won't win anything either! Now look at the section for your machine, and as usual, don't forget to SAVE this section of program ready for the second half. You won't be able to RUN the program successfully at this stage, although on some machines, you will see the basic graphics.

## SETTING UP THE GRAPHICS

Lines 12Ø to 14Ø hold the DATA for the coloured fruit. To economize on memory, the colour instructions are entered directly by control codes as explained on page 775—and the listings appear in colour. To enter these lines, go into extended mode (CAPS SHIFT and SYMBOL SHIFT), then press the appropriate colour key. Then enter graphics mode (CAPS SHIFT and 9), followed by the letters in the listing.

```
10 LET HFLAG=Ø: POKE 23658,8: RESTORE
   2Ø: GOSUB 84Ø
2Ø DATA 14,31,31,31,31,15,3,1,56,252,252,
   252,252,248,224,192
3Ø DATA 49,42,51,42,5Ø,255,255,255,152,84,
   216,84,84,255,255,255
4Ø DATA 3,4,8,28,62,62,62,28,28,19Ø,125,62,
   28,Ø,Ø,Ø
5Ø DATA Ø,Ø,16,28,15,7,3,Ø,4,12,26,56,248,
   24Ø,224,Ø
6Ø DATA Ø,Ø,7,15,31,31,15,7,8,248,24Ø,24Ø,
   224,224,192,128
7Ø DATA 1,3,3,7,15,15,24,1,128,192,192,224,
   24Ø,24Ø,24,128
8Ø DATA 1,7,15,31,31,31,31,15,224,24Ø,24Ø,
   24Ø,24Ø,224,192,128
9Ø DATA 8,8,8,8,73,42,28,8,16,56,84,146,16,
   16,16,16
1ØØ FOR i=USR "a" TO USR "P"+7: READ
   a: POKE i,a: NEXT i
11Ø LET TOTAL=1ØØ
12Ø DATA "AB", "IJ", "GH", "CD", "IJ",
   "AB", "EF", "KL", "MN", "EF", "GH",
   "MN"
13Ø DATA "IJ", "AB", "KL", "MN", "EF",
   "GH", "DD", "EF", "AB", "CD", "MN"
14Ø DATA "EF", "MN", "GH", "KL", "CD",
   "AB", "IJ", "AB", "IJ", "CD", "IJ",
   "MN"
15Ø DIM A$ (24,4): FOR i=1 TO 12 READ
   A$ (i): LET A$ (i+12)=A$(i) NEXT i
16Ø DIM B$ (24,4): FOR i=1 TO 12 READ
   B$ (i): LET B$(i+12)=B$ (i) NEXT i
17Ø DIM C$ (24,4): FOR i=1 TO 12 READ
   C$=(i): LET C$ (i+12)=C$(i) NEXT i
84Ø BORDER 7: PAPER 9: INK 2: CLS
85Ø PRINT ""
86Ø FOR i=Ø TO 1: FOR j=Ø TO 31: PRINT
   PAPER 7; AT i,j;"□": NEXT j: NEXT i
87Ø FOR i=2 TO 5: PRINT AT i,Ø; PAPER 6;
   "□□□□"; AT i,28;"□□□□":
   NEXT i
88Ø PRINT PAPER 6; INK 2;AT Ø,8;"1Ø
   PENCE A GAME"
89Ø PRINT PAPER 6; INK 2;AT
   1,4;"□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□";AT 2,4;"
   □□□▛▘▌▌▛▌▛▘▛▌
   ▛▘▛▌▌▌▌▌▜▘□□"
9ØØ PRINT PAPER 6; INK 2;AT
   1,4;"□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□";AT 3,4;"
   □□□▀▌▌▌▛▘▛□▛▗
   ▛□▛▗▌▌▌▐□□□"
91Ø PRINT PAPER 6; INK 2;AT
   1,4;"□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□";AT 4,4;
   "□□□▀▘▀▘▘□▀▘▘▘
   ▘□▘▘▀▘▘▝□□□"
92Ø PRINT PAPER 6; INK 2;AT
   1,4;"□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□";AT 5,4;
   "□□□□"; INK Ø;"██████
   ██████████□□□
   □"
93Ø FOR i=6 TO 14: FOR j=8 TO 23 STEP
   5: PRINT INK Ø;AT i,j;"    ": NEXT j: NEXT i
94Ø FOR i=8 TO 23: PRINT AT 15, i; INK
   Ø;"█": NEXT i
95Ø FOR I=6 TO 21: PRINT PAPER
   6;AT I,Ø;"□□□□□□□□";AT
   I,24;"□□□□□□□□": NEXT I
96Ø FOR I=16 TO 21: PRINT PAPER 6;AT I,8;
   "□□□□□□□□□□□□□□
   □□": NEXT I
97Ø PRINT INK 1; PAPER 8;AT 2,2;"WIN";AT
   2,27;"WIN";AT 3,2;"▀ ▀ ▀";AT
   3,27;"▀ ▀ ▀"
98Ø PRINT INK 1; PAPER 8;AT 4,2;"1Ø";AT
   4,27;"1ØØ";AT 6,2;"2Ø";AT 8,2;"5Ø";AT
   7,25;"JACKPOT";AT 8,25;"▀ ▀ ▀ ▀
   ▀ ▀ ▀"
99Ø PRINT INK 1; PAPER 8;AT 2Ø,3;"PRESS
   SPACE—TO SPIN REELS";AT 21,1;"OR
   HOLD OR NUDGE KEYS WHEN LIT"
1ØØØ PRINT AT 17,7;"□□□ 1 □□□□ 2
   □□□□ 3 □□□□"; AT 16,9; "█
   ███□████□██
   ██"
1Ø1Ø PRINT AT 18,7;"□4=1 & 2□5=2 &
   3□6=1 & 3";AT 19,7; INK 7; PAPER 2;
   "□HOLD KEYS 1 to 6□"
1Ø2Ø RETURN
```

Line 1Ø jumps to the subroutine at Line 86Ø which draws the fruit machine. All the instructions the player needs are displayed on the machine's front.

Line 1ØØ sets up the UDGs by READing from the DATA in Lines 2Ø to 9Ø. The UDGs, when set up, appear in Lines 12Ø to 17Ø as DATA ready for the reels to be set up. a$, b$ and c$ are the three reels, and the order of the fruit on the reels is determined by the order of the DATA.

## THE REEL THING

```
18Ø GOSUB 1Ø3Ø
1Ø3Ø PRINT A 5,Ø;A$(4): FOR X=Ø TO 3
   STEP 2: PRINT AT7,X; A$(4); "□": NEXT
   X: FOR X=Ø TO 5 STEP 2: PRINT AT 9,X;
   A$(4): NEXT X: FOR X=Ø TO 5 STEP 2:
   PRINT AT 1Ø,X; A$(1); "□": NEXT X
1Ø4Ø PRINT AT 11,Ø;C$(2);C$(2);C$(2);AT
   12,Ø;B$(1);B$(1);B$(1)
1Ø5Ø PRINT AT 5,26;C$(1);C$(1);C$(1);AT
   6,26;C$(3);C$(3);C$(3);AT 9,26;C$(4);
   C$(4);C$(4)
```




```
1060 PRINT AT 7,10;A$(1);AT 10,10;A$(2);
   AT 13,10;A$(3)
1070 PRINT AT 7,15;B$(1);AT 10,15;B$(2);
   AT 13,15;B$(3)
1080 PRINT AT 7,20;C$(1);AT 10,20;C$(2);
   AT 13,20;C$(3)
1090 PRINT AT 16,26; INK 2; PAPER 6;
   "NUDGE";AT 17,26;"KEYS";AT15,25;
   "Q—W—E▜";AT 18,25;"A—S—D◙"
1100 PRINT INK 0; PAPER 7;AT 12,25; "▗
   ▄▄▄▄▄▖"; AT 13,25; "▐";
   AT 13,31; "▌"; AT 14,25; "▝▀▀▀
   ▀▀▘"; AT 13,26; INK 2; "███
   ██"
1110 PRINT PAPER 6; INK 0; INVERSE 1; AT
   15,0; "TOTAL"; INVERSE 0; AT 17,0; "£";
   AT 18,0; "p"; AT 17,1; PAPER 7; BRIGHT
   1;"□□□"; AT 18,1; TOTAL
1120 PRINT #1; AT 0,0; "□□□□ YOU
   START WITH ONE POUND"
1130 PAUSE 0
1140 PRINT #1;AT 0,0;"□□□□□□□
   □□□□□□□□□□□□□□
   □□□□□□□□□□"
1150 RETURN
```

The subroutine starting at Line 1030 draws the reels at their starting positions and completes some of the display details. The player is told that he starts with one dollar—each spin costs ten cents.

## INITIALIZATION

```
40 POKE53280,0:POKE53281,0:A$=
```

```
   "□PLEASE":B$="□WAIT!□":C$=
   "■■■■■■■■"
45 PRINT"◻■■■■■■■■■":
   FORA=1TO7:FORB=1TO13:PRINT"◻";
   MID$(C$,A,1);SPC(B);A$;
50 PRINTTAB(33-B);B$:NEXTB:PRINT:
   NEXTA
55 POKE52,48:POKE56,48:CLR:POKE56334,
   PEEK(56334)AND254:POKE1,PEEK(1)AND
   251
60 FORA=0TO1023:POKEA+12288,PEEK
   (A+53248):NEXT:POKE1,PEEK(1)OR4
65 POKE56334,PEEK(56334)OR1:POKE53280,
   8:FORA=0TO655:READB:POKE12288+A,
   B:NEXTA
10000 DATA255,213,253,253,253,253,213,255
10005 DATA255,127,127,127,127,127,87,255
10010 DATA255,213,255,213,213,215,213,255
10015 DATA255,87,215,87,87,255,87,255
10020 DATA255,213,255,253,253,255,213,255
10025 DATA255,87,215,87,87,215,87,255
10030 DATA255,215,215,213,213,255,255,255
10035 DATA255,215,215,87,87,215,215,255
10040 DATA255,213,215,213,213,255,213,255
10045 DATA255,87,255,87,87,215,87,255
10050 DATA255,204,0,240,252,204,204,240
10055 DATA255,204,0,48,252,204,204,252
10060 DATA252,204,0,240,252,204,204,240
10065 DATA204,204,252,240,0,204,255,0
10070 DATA204,204,204,204,0,204,255,0
10075 DATA204,204,204,204,0,204,252,0
10080 DATA0,0,0,3,3,15,15,0
10085 DATA48,252,252,255,255,255,255,0
10090 DATA0,0,0,0,0,192,192,0
10095 DATA15,15,3,0,0,0,0,0
10100 DATA255,255,255,252,48,63,15,0
10105 DATA192,192,0,0,0,0,0,0
10110 DATA0,3,15,15,15,15,15,15
10115 DATA0,255,243,252,252,252,252,252
10120 DATA0,0,192,192,192,192,192,192
10125 DATA15,15,15,3,3,0,0,0
10130 DATA252,252,252,243,255,252,0,0
10135 DATA 192,192,192,0,0,0,0,0
10140 DATA0,0,10,34,170,136,170,162
10145 DATA12,48,254,186,42,162,170,34
10150 DATA0,0,128,32,168,40,168,136
10155 DATA170,40,10,2,0,0,0,0
10160 DATA0,0,0,0,0,0,0,0
10165 DATA168,138,170,34,168,168,32,0
10170 DATA102,102,102,0,0,0,0,0
10175 DATA168,160,128,0,0,0,0,0
10180 DATA0,0,0,0,0,3,3,3
10185 DATA4,16,84,252,252,255,255,255
10190 DATA0,0,0,0,0,0,0,0
10195 DATA3,15,15,15,15,15,3,0
10200 DATA255,255,255,255,255,255,255,0
10205 DATA0,192,192,192,192,192,0,0
10210 DATA0,0,0,0,1,1,1,1
10215 DATA16,16,84,68,81,81,81,81
10220 DATA0,0,0,0,0,0,0,0
10225 DATA1,5,21,85,0,0,0,0
10230 DATA81,84,85,85,16,0,0,0
10235 DATA0,64,16,84,0,0,0,0
10240 DATA252,204,204,204,204,204,
   252,0
10245 DATA240,48,48,48,48,48,252,0
10250 DATA252,12,12,252,192,192,252,0
10255 DATA252,12,12,60,12,12,252,0
10260 DATA204,204,204,252,12,12,12,0
10265 DATA252,192,192,252,12,12,252,0
10270 DATA192,192,192,252,204,204,252,0
10275 DATA252,12,12,12,12,12,12,0
10280 DATA252,204,204,252,204,204,252,0
10285 DATA252,204,204,252,12,12,12,0
10290 DATA0,0,0,0,10,32,40,42
10295 DATA12,48,48,240,12,130,130,130
10300 DATA0,0,0,0,160,8,136,168
10305 DATA10,0,0,0,0,0,0,0
10310 DATA0,40,130,162,170,40,0,0
10315 DATA160,0,0,0,0,0,0,0
10320 DATA252,192,192,192,192,192,252,0
10325 DATA240,204,204,204,204,204,240,0
10330 DATA252,192,192,240,192,192,252,0
10335 DATA252,192,192,240,192,192,192,0
10340 DATA48,204,192,192,204,204,60,0
10345 DATA204,204,204,252,204,204,204,0
10350 DATA252,48,48,48,48,48,252,0
10355 DATA192,192,192,192,192,192,252,0
10360 DATA192,240,252,204,204,204,
   204,0
10365 DATA48,204,204,204,204,204,48,0
10370 DATA240,204,204,240,192,192,192,0
10375 DATA240,204,204,240,204,204,
   204,0
10380 DATA48,204,192,48,12,204,48,0
10385 DATA252,48,48,48,48,48,48,0
10390 DATA204,204,204,204,204,204,48,0
10395 DATA12,48,48,252,48,240,204,0
10400 DATA0,0,0,60,60,0,0,0
10405 DATA255,255,255,255,255,255,255,255
```

Lines 40 to 50 set up a display asking the player to wait. The Commodore 64 takes some time to set up the UDGs needed for the fruit.

Lines 55 to 65 READ the DATA from the end of the program—see Lines 10000 to 10405—to set up the UDGs.

## INSTRUCTIONS

```
70 POKE53282,7:POKE53283,2:POKE53280,7:
   POKE53281,7
75 PRINT"◻■";CHR$(14);SPC(15);"■
   ◪□⊟⊟⊟⊟◪◪◫□"
80 PRINT"■■◫OU HAVE £1 TO START
   WITH. ◪T IS 10P A"
85 PRINT"GAME AND YOU PLAY UNTIL YOU
   HAVE NO"
90 PRINT"MONEY LEFT.":PRINT"■■⊟
   □◪◫⊟□□■:-"
95 PRINT"■■<■□■⊟⊟>",
   "-■PIN REELS/GAMBLES"
100 PRINT"<1>","-⊟ANCELS HOLDS"
105 PRINT"<2>","-◫OLD LEFT REEL"
110 PRINT"<3>","-◫OLD MIDDLE
   REEL"
115 PRINT"<4>","◫OLD RIGHT REEL"
120 PRINT"<5>","-◪UDGE LEFT REEL
   ◪□"
125 PRINT"<6>","-◪UDGE MIDDLE
   REEL ◪□"
130 PRINT"<7>","-◪UDGE RIGHT
   REEL ◪□"
135 PRINT"<8>","-◪UDGE LEFT REEL
   ⊟□◘◪"
140 PRINT"<9>","-◪UDGE MIDDLE
   REEL ⊟□◘◪"
145 PRINT"<0>","-◪UDGE RIGHT
   REEL ⊟□◘◪"
150 PRINT"<⊟⊟◫◪⊟◪>",
   "-⊟OLLECT WIN"
155 PRINT"■■■■□LEASE PRESS THE
   ■PACE-BAR TO PLAY ..."
160 GETA$:IFA$<>"□"THEN160
```

Lines 70 to 160 set up the instructions screen.

## PREPARING THE GAME

```
165 PRINT"◻";CHR$(142):POKE53280,10:
   POKE53281,10:POKE53272,29
170 POKE53270,PEEK(53270)OR16:POKE
   53282,7:POKE53283,2:IFXX=1THEN190
175 DIMR1%(15),R2%(15),R3%(15),W%(9),
   F$(6)
180 FORA=0TO15:READR1%(A),R2%(A),
   R3%(A):NEXT:FORA=0TO9:READW%
   (A):NEXTA
185 FORA=0TO6:READF$(A):NEXTA
190 POKE53280,13:POKE53281,1:PRINT
   "◻■",F$(3);"◘■";F$(3);"◘■";
   F$(3);
195 PRINT"◘■■■■■□2□00":
   PRINT"■",F$(2);"◘■";F$(2);
   "◘■";F$(2);
200 PRINT"◘■■■■■□1□50":
   PRINT"■",F$(5);"◘■";F$(5);
   "◘■";F$(5);
205 PRINT"◘■■■■■□1□00":
   PRINT"■",F$(4);"◘■";F$(4);
   "◘■";F$(4);
210 PRINT"◘■■■■■□0□80":
   PRINT"■",F$(6);"◘■";F$(6);
   "◘■";F$(6);
215 PRINT"◘■■■■■□0□60":
   PRINT"■",F$(0);"◘■";F$(0);
   "◘■";F$(0);
220 PRINT"◘■■■■■□0□40":
   PRINT"■",F$(1);"◘■";F$(1);
   "◘■";F$(1);
225 PRINT"◘■■■■■□0□30":
   PRINT"■",F$(0);"◘■";F$(0);"◘
   ■■□□□■";
```







**Colourful graphics adorn SUPERFRUIT on the Acorn** **.....and SUPERFRUIT on the Spectrum**

```
230 PRINT"■■■■■■□Ø□3Ø":
    PRINT"■",F$(1);"■■";F$(1);"■
    ■■□□□■";
235 PRINT"■■■■■■□Ø□2Ø":
    PRINT"■",F$(1);"■■■□□□
    ■□□□■■■■□Ø□1Ø"
240 PRINT"■■■■■■■■■■■
    ■■■■■■■■■□□□□
    ■■■■■■□";
245 GETA$:IFA$< >CHR$(13)THEN245
1Ø41Ø DATAØ,1,2,3,5,6,6,2,Ø,5,3,4,4,6,5,6,4,
    3,1,2,Ø,3,Ø,5,2,1,4,6,5,1,Ø,6,6
1Ø415 DATA1,4,3,2,Ø,1,5,3,2,4,6,6,6,4,5
1Ø42Ø DATA2ØØ,15Ø,1ØØ,8Ø,6Ø,4Ø,3Ø,2Ø,1Ø,
    Ø
1Ø425 DATA"■*+,■■■■■-./",
    "■|:;<■■■■■■=>?","■|PQR■
    ■■■■STU","■JKL■■■■■■MN
    O"
1Ø43Ø DATA"■|£]↑■■■■■■←!#",
    "■VWX■■■■■■YZ[","■|$%&■
    ■■■■■'()"
```

Lines 17Ø to 185 prepare the reels. The arrays DIMensioned in Line 175 are used as follows: R1%, R2% and R3% contain the arrangement of the fruit on each of the reels, W% are the wins from each of the win lines, and F$ the information necessary to display the fruit. The arrays are filled with DATA READ from Lines 1Ø41Ø to 1Ø43Ø.

Lines 19Ø to 245 set up the second screen the player sees—the winning line information.

## SETTING UP THE MACHINE

```
250 POKE5328Ø,Ø:POKE53281,1:PRINT
    "■■|";:FORA=1TO24
255 PRINT"●●●●●●●●●●●●
    ●●●●●●●●●●●●●●
    ●●●●●●●●●●●●●●
    ●●●●";:NEXT:FORA=1984TO2Ø23
260 POKEA,81:POKEA+54272,13:NEXTA:
    PRINT"■■","■■■■■□□
    ■■■■■■■■■■□"
265 PRINT"■■■■";:FORA=ØTO9:
    PRINT"■■■■■■■■□□□□
    □■■□□□□□□■■□□□□
    □■■■□";
270 A$=RIGHT$("ØØ"+MID$(STR$(W%
    (A)),2),3):PRINTLEFT$(A$,1);"□";
    RIGHT$(A$,2);
275 PRINT"■■■■■■■■";:NEXTA
280 PRINT"■■■■■■■■■■□
    ■■■●■■□■■■●■■
    □■■■":
285 PRINT"■■■■■■■■■■■
    ■■■■■■■■■■■■■□■
    ●■■■■@A■BC■DE■FG■
    HI■■■■■■■■■■■■";
290 PRINT"■■■■■■■■■■■
    ■■■■■■■■■●■■■■
    □■";
295 M=RND(-TI):M=1ØØ:H%=-1:I%=
    -1:J%=-1:P%=INT(RND(1)*16):Q%
    =INT(RND(1)*16)
300 R%=INT(RND(1)*16):N%=Ø
305 GOSUB95ØØ
```

Lines 25Ø to 3Ø5 draw the fruit machine with windows ready for the reels, and the nudge, gamble and money areas.

## GRAPHICS AND SOUND

Don't type in Line 1Ø until you are sure the program is finished and debugged.

```
10 ONERRORRUN
20 VDU23,224,1,3,7,7,15,15,15,31,23,225,
    128,192,96,16Ø,176,2Ø8,2Ø8,216,23,226,
    31,63,63,63,63,3,1,Ø,23,227,232,252,252,
    252,252,192,128,Ø
30 VDU23,228,Ø,Ø,Ø,3,7,14,31,15,23,229,Ø,Ø,
    Ø,Ø,128,192,96,192,23,23Ø,7,27,6Ø,118,
    251,126,6Ø,24,Ø,23,231,152,6Ø,118,251,
    126,6Ø,24,Ø,23,232,15,248,128,Ø,Ø,Ø,Ø,Ø
40 VDU23,233,Ø,1,1,3,3,7,7,15,23,234,128,
    192,192,224,224,24Ø,24Ø,248,23,235,15,Ø,
    15,7,3,3,1,Ø,Ø,23,236,248,Ø,248,24Ø,224,
    192,136,24Ø
50 VDU23,237,255,Ø,255,255,Ø,225,146,227,
    23,238,255,Ø,255,255,Ø,142,73,2Ø6,23,
    239,146,226,Ø,255,255,Ø,255,Ø,23,24Ø,73,
    73,Ø,255,255,Ø,255,Ø
60 VDU23,241,Ø,2,15,27,3Ø,55,63,23,23,242,
    Ø,8,6Ø,246,254,223,123,238,23,243,31,13,
    7,7,2,3,1,Ø,23,244,19Ø,236,12Ø,248,2Ø8,
    24Ø,224,192,23,245,32,64,192,Ø,Ø,Ø,Ø,Ø
70 VDU23,246,Ø,Ø,Ø,Ø,3,3,7,7,23,247,Ø,Ø,Ø,
    Ø,24Ø,216,216,22Ø,23,248,15,15,15,7,7,3,
    3,1,23,249,236,252,252,248,24Ø,24Ø,224,
    192,23,25Ø,12,112,64,24Ø,Ø,Ø,Ø,Ø
80 VDU23,251,Ø,Ø,Ø,Ø,Ø,1,3,3,23,252,Ø,Ø,Ø,
    192,192,224,16Ø,176,23,253,7,7,15,15,31,
    31,15,7,23,254,2Ø8,2Ø8,248,248,252,252,
    248,24Ø,23,255,28,112,192,Ø,Ø,Ø,Ø,Ø
90 *FX9,5
100 *FX1Ø,5
110 *FX11,Ø
120 A%=RND(-TIME):@%=Ø:ENVELOPE
    1,1,1,1,Ø,5,5,Ø,126,Ø,Ø,-8,126,Ø
```

Each of the lines contains instructions for drawing the fruit graphics in a VDU statement. Line 2Ø is the bell; Line 3Ø, the cherry; Line 4Ø, the acorn; Line 5Ø, the bar; Line 6Ø, the strawberry; Line 7Ø, the blueberry; and Line 8Ø, the pear. Lines 9Ø and 1ØØ set the flash rate, while Line 11Ø turns off the auto repeat.

Line 12Ø chooses a random seed, and sets each of the 14 parameters needed for the ENVELOPE for the blip.

## INSTRUCTIONS

```
130 MODE6:VDU23;8202;Ø;Ø;Ø;:PRINT'TAB
    (13)"SUPERFRUIT"'
140 PRINTTAB(3)"WELCOME TO THE BBC
    FRUIT MACHINE !"
150 PRINT''"□You are given £1 to start with. It
    costs 1Øp per game to play and you play
    until you run out of money."
160 PRINT''"Controls :"''"<SPACE>□□—
    □Spin reels/gambles"''"<1>□□□□
    □□—□Cancel Hold"''"<2>□□□
    □□□—□Hold left reel"
170 PRINT"<3>□□□□□□□—□Hold
    middle reel"''"<4>□□□□□□□—□
    Hold right reel"''"<5>□□□□□□□
    —□Nudge left reel up"
180 PRINT"<6>□□□□□□□—□
    Nudge middle reel up"''"<7>□□□□
    □□—□Nudge right reel up"''"<8>□
    □□□□□□—□Nudge left reel down"
190 PRINT"<9>□□□□□□□—□
    Nudge middle reel down"''"<Ø>□□□
    □□□—□Nudge right reel down"'
    "<RETURN>□—□Collect winnings"
200 PRINTTAB(4,24)"Press the space-bar to
    continue . . .";:*FX15,1
```

Lines 14Ø to 2ØØ set up the first screen the player sees—the instructions that are needed for playing the game.

## PREPARING THE GAME

The symbol ▁ denotes the underline character:

```
210 REPEATUNTILGET=32:MODE2:VDU19;7;
    Ø;19,7;Ø;Ø,23;8202;Ø;Ø;Ø;19,8,1;Ø;19,9,1;
    Ø;19,1Ø,1;Ø;19,11,1;Ø;19,12,1;Ø;19,13,9;Ø;
    19,15,12;Ø;
220 DIMF$(6),R1%(15),R2%(15),R3%(15),
    W%(9)
230 FORA%=ØTO6:REPEAT:READB%:F$(A%)
    =F$(A%)+CHR$B%:UNTILB%=13:F$
    (A%)=LEFT$(F$(A%),LEN(F$(A%))−1):
    NEXT
240 DATA18,Ø,7,237,238,8,8,1Ø,239,24Ø,13,
    18,Ø,2,233,234,8,8,1Ø,235,236,13,18,Ø,4,
    246,247,8,8,1Ø,248,249,8,11,18,Ø,5,25Ø,
    13,18,Ø,1,241,242,8,8,1Ø,243,244,8,11,18,
    Ø,2,245,13
250 DATA18,Ø,2,251,252,8,8,1Ø,253,254,8,11,
    18,Ø,7,255,13,18,Ø,3,224,225,8,8,1Ø,226,
    227,13,18,Ø,1,228,229,8,8,1Ø,23Ø,231,8,
    11,18,Ø,2,232,13
260 FORA%=ØTO15:READR1%(A%),R2%
    (A%),R3%(A%):NEXT
270 DATAØ,1,2,3,5,6,6,2,Ø,5,3,4,4,6,5,6,4,3,1,
    2,Ø,3,Ø,5,2,1,4,6,5,1,Ø,6,6,1,4,3,2,Ø,1,5,3,
    2,4,6,6,6,4,5
280 FORA%=ØTO9:READW%(A%):NEXT
290 DATA2ØØ,15Ø,1ØØ,8Ø,6Ø,4Ø,3Ø,2Ø,1Ø,Ø
300 PROCwinlines
1040 DEFPROCwinlines:VDU5:MOVE64,992:
    PRINTSTRING$(3,F$(Ø)+
    CHR$11+CHR$32);:GCOLØ,6:
    PRINT"▁□£2.ØØ"
1050 MOVE64,896:PRINTSTRING$(3,F$(1)+
    CHR$11+CHR$32);:GCOLØ,6:PRINT
    "▁□£1.5Ø":MOVE64,8ØØ:PRINT
    STRING$(3,F$(2)+CHR$32);:GCOLØ,6:
    PRINT"▁□£1.ØØ"
1060 MOVE64,7Ø4:PRINTSTRING$(3,F$(3)+
    CHR$32);:GCOLØ,6:PRINT"▁□£Ø.8Ø":
    MOVE64,6Ø8:PRINTSTRING$(3,F$(4)+
    CHR$32);:GCOLØ,6:PRINT"▁□£Ø.6Ø"
1070 MOVE64,512:PRINTSTRING$(3,F$(5)+
    CHR$32+CHR$11);:GCOLØ,6:PRINT
    "▁□£Ø.4Ø":MOVE64,416:PRINT
    STRING$(3,F$(6)+CHR$32);:GCOLØ,6:
    PRINT"▁□£Ø.3Ø"
1080 MOVE64,32Ø:PRINTSTRING$(2,F$(5)+
    CHR$32+CHR$11);"▁□□";:GCOLØ,
    6:PRINT"▁□£Ø.3Ø":MOVE64,224:PRINT
    STRING$(2,F$(6)+CHR$32);"▁□□";:
    GCOLØ,6:PRINT"▁□£Ø.2Ø"
1090 MOVE64,128:PRINTF$(6);
    "□▁□□▁□□";:GCOLØ,6:PRINT
    "▁□£Ø.1Ø":GCOLØ,15:MOVE64,32:
    PRINT"□□Space to start"
1100 *FX15,1
1110 REPEATUNTILGET=32:VDU4:CLS:
    ENDPROC
```

After setting up the graphics mode in Line 21Ø, four arrays are DIMensioned.

The next section of program contains DATA for the arrays. F$ contains the information needed to colour each fruit—they are PRINTed after a VDU 5, with the colour set by GCOLØ, followed by the colour number, or in character codes. R1%, R2%, and R3% contain the contents of the reels, each number corresponding to a fruit. The final array, W%, contains the amounts paid out, in ascending order.

Line 3ØØ calls PROCwinlines—to be found at Lines 1Ø4Ø to 111Ø—which sets up a display telling the player what the winning combinations are.

## DRAWING THE MACHINE

```
310 COLOUR13Ø:CLS:COLOUR128:VDU28,3,
    15,6,4,12,28,8,15,11,4,12,28,13,15,16,4,
    12,26:COLOUR13Ø:COLOUR3:PRINTTAB
    (5,1);"SUPERFRUIT"
320 VDU5:FORA%=1TO5:VDU29,
    224+(A%−1)*96;224;:MOVEØ,Ø:GCOL
    Ø,7+A%:FORA=ØTO2*PI□STEPPI/15:
    MOVEØ,Ø:PLOT85,3Ø*SINA,3Ø*COSA:
    NEXT:MOVE−32,8:GCOLØ,Ø:PRINTA%:
    NEXT
330 VDU29,Ø;Ø;:GCOLØ,4:FORA%=ØTO2:
    MOVE192+A%*32Ø,384:PROChbox:NEXT
340 M%=1ØØ:H%=TRUE:I%=TRUE:J%=
    TRUE:P%=RND(16)−1:Q%=RND(16
    −1:R%=RND(16)−1:N%=Ø
350 @%=&2Ø2ØA:GCOLØ,4:MOVE256,16Ø:
    PRINT"NUDGE":MOVE7Ø4,152:GCOLØ,1:
    PRINT"GAMBLE":GCOL3,4:MOVE256,6Ø:
    PRINT"Credit□£";M%/1ØØ
```

Line 31Ø clears the screen to green, sets up the white reels, and PRINTs the title. The nudge lights are drawn in Line 32Ø, while the blue hold boxes are drawn in Line 33Ø.

Line 34Ø initializes a series of variables. H%, I%, and J% are hold flags for the reels—FALSE if held, TRUE if free. M% is money; P%, Q% and R% are pointers to each reel, and N% is the number of nudges available.

Finally, in this section of program, Line 35Ø sets the format—two decimal places—of the money display.

If you RUN the program at this stage it will stop with an error in Line 33Ø as PROChbox is undefined so far.

## SETTING UP THE GRAPHICS

```
10 PMODE3,1:PCLS:CLS
20 DIM B(12),C(12),A(12),BR(12),S(12),
   PL(12),P(12),R1(15),R2(15),R3(15),W(9),
   H(29)
30 DRAW"BM16,ØC2L2GLG4DG4D2R7FRFR3
   ERER7U2H4UH4LH":PAINT(14,1Ø):DRAW
   "BM17,2C1F4DF"
40 GET(Ø,Ø)−(31,15),B,G
50 DRAW"BM62,ØC2L6GL6G2C4L3GLGLGFD
   GLGLGLGFRFRFR3ERERER5FRFR3ERERE
   HLHLHL5HEHLHLH":PAINT(48,8):DRAW
   "BM41,8C1FRFRFRNFUR2UR2URBM−4,
   −2HBM−5,7HBR17H"
60 GET(32,Ø)−(63,15),C,G
70 DRAW"BM8Ø,ØC3G8R17NH8BD2LNL15GL
   GLGLGL3NH3RFR7E":PAINT(8Ø,4):PAINT
   (8Ø,13)
80 GET(64,Ø)−(95,15),A,G
90 DRAW"BM96,ØC4R3ØBD2L3ØDR3ØBD6U
   BU2HL4D2NR4D2BL7U3HL3GDNR4D2BL1
   2R4EHEHL5D2NR2D2BD2R3ØDL3ØBD2R3
   Ø"
100 GET(96,Ø)−(127,15),BR,G
110 DRAW"BM148,ØC2GL3GC4LHLGL3G4RF
    3RF5RFR3ERE4UER2E3LH3LHLGL5GL":
    PAINT(144,8):DRAW"BM138,3C1RBR13RB
    DBL1ØLBDBL5LBDBR8RBR9RBDBL4LBL
    13LBDBR1ØRBR4BDLBL9LBDBR4RBG2LB
    R6R"
120 GET(128,Ø)−(159,15),S,G
130 DRAW"BM186,ØC3L3L2GL5DGR6DF4DG2L
    G3LGL5H6E5R2":PAINT(176,8):DRAW"BR
    6BDC1D2F"
```

```
14Ø GET(16Ø,Ø)-(191,15),PL,G
15Ø DRAW"BM218,ØC2L4DL3D3F4DF4LGLGL
    13HLHUE7REU2":PAINT(2Ø8,8):DRAW
    "BM211,6C1DF2"
16Ø GET(192,Ø)-(223,15),P,G
```

The Fruit Machine is drawn in PMODE3, and the GET and PUT commands are used to display the fruit symbols on the reels. The arrays needed for the fruit symbols are DIMensioned in Line 2Ø—B for the bell, C, for the cherry, A for the acorn, BR for the bar, S for the strawberry, PL for the plum, and P for the pear. R1, R2 and R3 are the contents of the three reels; W contains the win amounts; and H is used for holding the reels.

Each pair of lines from Line 3Ø to Line 16Ø DRAW the fruit and then GET them into the appropriate array.

## INSTRUCTIONS

```
17Ø B$=CHR$(128):CLS:PRINT@9,
    B$"superfruit"B$
18Ø PRINT" YOU HAVE $1 TO START WITH.IT
    IS 1ØC A GAME AND YOU PLAY UNTIL □
    □ □ YOU HAVE NO MONEY.";
19Ø PRINT"controls:—":PRINT"<SPACE>"
    TAB(8)"—SPIN REELS/GAMBLES":PRINT
    "<1>"TAB(8)"—CANCELS HOLDS"
2ØØ PRINT"<2>"TAB(8)"—HOLD
    LEFT REEL":PRINT"<3>"TAB(8)
    "—HOLD MIDDLE REEL":
    PRINT"<4>"TAB(8)"—HOLD RIGHT
    REEL"
21Ø PRINT"<5>"TAB(8)"—NUDGE
    LEFT REEL UP":PRINT"<6>"
    TAB(8)"—NUDGE MIDDLE REEL UP":
    PRINT"<7>"TAB(8)"—NUDGE RIGHT
    REEL UP"
22Ø PRINT"<8>"TAB(8)"—NUDGE
    LEFT REEL DOWN":PRINT"<9>"
    TAB(8)"—NUDGE MIDDLE REEL
    DOWN":PRINT"<Ø>"TAB(8)
    "—NUDGE RIGHT REEL DOWN":
    PRINT"<ENTER>"TAB(8)
    "—COLLECT WIN";
```

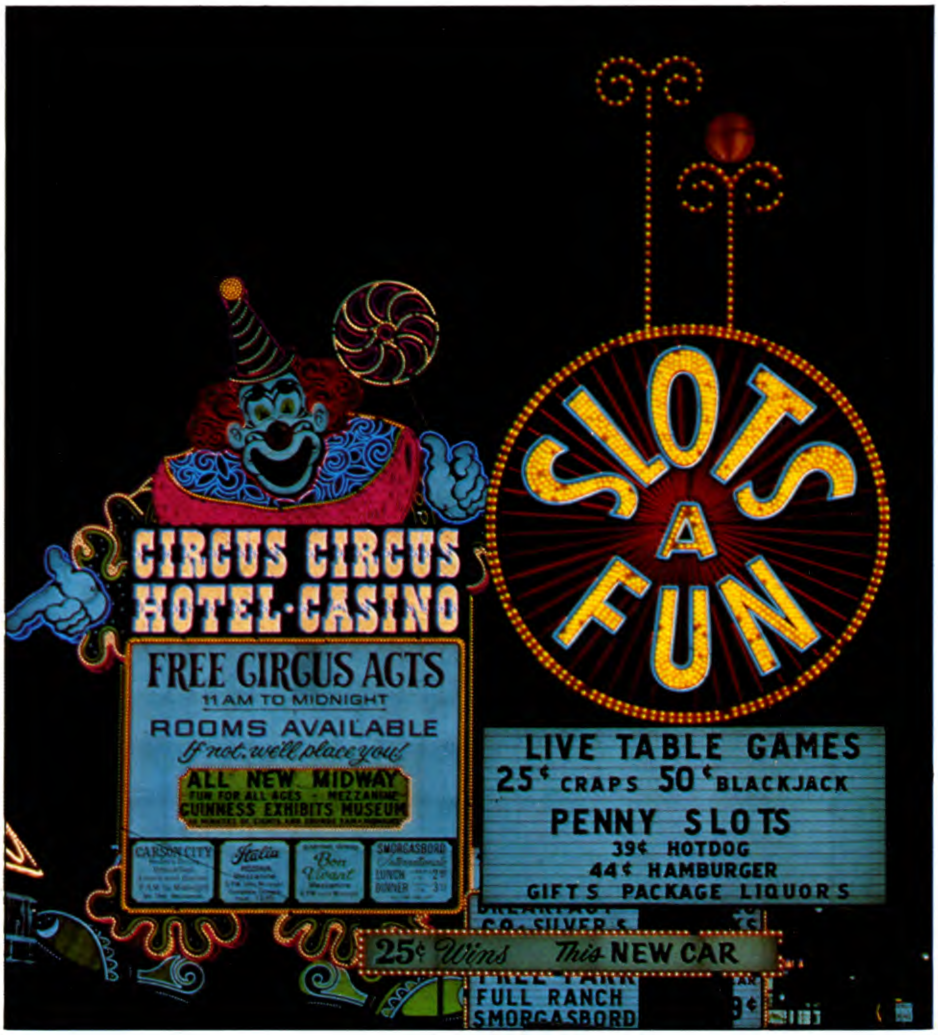
Lines 17Ø to 22Ø display the instructions on the text screen.

## PREPARING THE MACHINE

```
23Ø IF INKEY$<>"□" THEN 23Ø
24Ø FORA=ØTO15:READR1(A),R2(A),R3(A):
    NEXT
25Ø DATA Ø,1,2,3,5,6,6,2,Ø,5,3,4,4,6,5,6,4,3,
    1,2,Ø,3,Ø,5,2,1,4,6,5,1,Ø,6,6,1,4,3,2,Ø,1,5,
    3,2,4,6,6,6,4,5
26Ø FORA=ØTO9:READW(A):NEXT
27Ø DATA 2ØØ,15Ø,1ØØ,8Ø,6Ø,4Ø,3Ø,2Ø,1Ø,Ø
28Ø GOSUB4ØØØ
29Ø SCREEN1,Ø:PCLS3:DRAW"BM84,4C2S2Ø
    LDRDLBR2NU2RU2BRND2RDLBEBRNRD
    NRDRBRU2RDLFBRUNRURBRND2RDLFB
    RNU2RU2BRD2BR2U2LR2"
3ØØ FORK=ØTO2:LINE(4Ø+64*K,2Ø)
    -(87+64*K,115),PRESET,BF:NEXT
31Ø FORK=ØTO2:DRAW"BM"+STR$
    (4Ø+64*K)+",124S16R12D4L12U4BFD2
    BRUNLUBR2RD2LU2BR3D2RBR2U2S8RF
    D2GL":NEXT
32Ø GET(38,122)-(91,143),H,G
33Ø COLOR4:FORK=1TO5:LINE(1Ø+K*16,
    158)-(21+K*16,169),PSET,BF:NEXT
34Ø DRAW"BR3ØC1S24U2F2U2BRD2RU2BRD
    2S8RE2U2H2LS24BR3LD2RUBENRDNRD
    R"
35Ø GOTO 35Ø
4ØØØ CLS:PRINT@11,"winlines"
4Ø1Ø PRINT@65,"BAR□□□□BAR□□
    □□BAR":PRINT"□ACORN□□ACORN
    □□ACORN":PRINT"□PLUM□□□
    PLUM□□□PLUM"
4Ø2Ø PRINT"□STRWBY□STRWBY□
    STRWBY":PRINT"□PEAR□□□PEAR□
    □□PEAR":PRINT"□BELL□□□BELL
    □□□BELL"
4Ø3Ø PRINT"□CHERRY□CHERRY□
    CHERRY":PRINT"□BELL□□□BELL□
    □□□□-":PRINT"□CHERRY□
    CHERRY□□□-":PRINT"□CHERRY□
    □□-□□□□□□□-"
4Ø4Ø FORA=ØTO9:IF A<7 THENPRINT
    @89+A*32,USING"$$#.##";W(A)/
    1ØØ;:GOTO4Ø6Ø
4Ø5Ø PRINT@89+A*32,USING
    "$$#.##";W(A-1)/1ØØ
4Ø6Ø NEXT
4Ø7Ø PRINT@449,"PRESS SPACE TO
    CONTINUE"
4Ø8Ø IFINKEY$<>"□" THEN4Ø8Ø
4Ø9Ø RETURN
```

Lines 24Ø and 25Ø set up the reels—each number represents one of the fruit. Lines 26Ø and 27Ø set up the win values. Line 28Ø jumps to the subroutine starting at Line 4ØØØ which displays the winning lines and their values.

Lines 29Ø and 34Ø initialize the screen. Notice that Line 29Ø switches on the high resolution screen for the first time so the completed machine appears.



# CLIFFHANGER: SETTING THE SCENE

**You can't have a Cliffhanger without a cliff. It's now time to slip on the slope which Willie will have to scale, plus the sky above it and the land below**

The titles and credits have rolled. The overture has played. Now's the time to roll on the scenery—or in the case of Cliffhanger, *INPUT*'s computer game—to scroll on the scenery.

This is a fairly simple process. You already have the data which defines the profile of the slope. Above the slope is sky and below it is the land—and they are simply a matter of filling in colours. But then you have to scroll off the instruction page and scroll on the sky and slope.

The routine listed below scrolls on the scenery.

```
      org 58303
lsi   ld a,16
      ld (57328),a
      ld ix,58Ø34
      ld b,32
      push bc
      call scl
      ld a,Ø
      ld (57329),a
      ld a,(ix+Ø)
      dec ix
      cp 33
      jr nz,lv
      dec b
      ld a,(57328)
      dec a
      ld (57328),a
      ld a,1
      ld (57329),a
lv    ld a,(57328)
      ld b,a
      ld hl,31
      ld a,45
      call lg
      ld bc,57264
      ld a,(57329)
      cp 1
      jr nz,3
      ld bc,57272
      ld a,44
      call print
      ld a,(57328)
      ld b,a
      ld a,23
      sub b
      ld b,a
      ld a,32
      ld de,32
      add hl,de
      call lg
      pop bc
      djnz 175
      ld hl,49
      ld b,12
      ld a,41
      ld ix,57973
      call me
      ld hl,113
      ld b,7
      call me
      call elb
      ret
scl   ld hl,16384
      ld b,216
lpi   ld c,31
lpj   inc hl
      ld a,(hl)
      dec hl
      ld (hl),a
      inc hl
      dec c
      jr nz,lpj
      inc hl
      djnz lpi
      ret
lg    push bc
      ld bc,15616
      call print
      ld de,32
      add hl,de
      pop bc
      djnz lg
      ret
elb   ret
      org 58155
me    *
      org 58217
print *
```

And you need some extra data:

```
5 CLEAR 57ØØØ
1Ø FOR n=57973 TO 58Ø34
2Ø READ a: POKE n,a: PRINT n;"  ";CHR$ a
3Ø NEXT n
4Ø DATA 83,67,79,82,69,45,48,48,48,48,48,48,
   76,73,86,69,83,45,53,71,65,77,69,32,79,86,
   69,82,32,33,33,33,35,35,33,35,35,35,33,35,
   35,33,35,33,35,35,35,35,33,35,33,35,35,35,
   35,33,35,35,33,35,35,35
```

## SETTING THE SCENE

The Y coordinate—16—of the top right of the horizon is loaded in the first workspace location, memory location 57,328. Then the last byte of the slope profile data is loaded into the IX register pair. The last byte of the slope profile data is the one that defines the slope of the top right-hand end of the horizon, of course.

The B register is loaded with 32 so that it can be used as a counter to count across the 32 columns of the screen. This is then stored on the stack.

The **scl** routine is called. This is the routine that scrolls the screen to the left.

The second workspace, memory location 57,329, is going to be used as a flag to tell the routine whether the slope is level or it is going down. A Ø in this location means that the slope continues flat. A 1 means that it is going down. But to initialize it the contents of this location are set to zero.

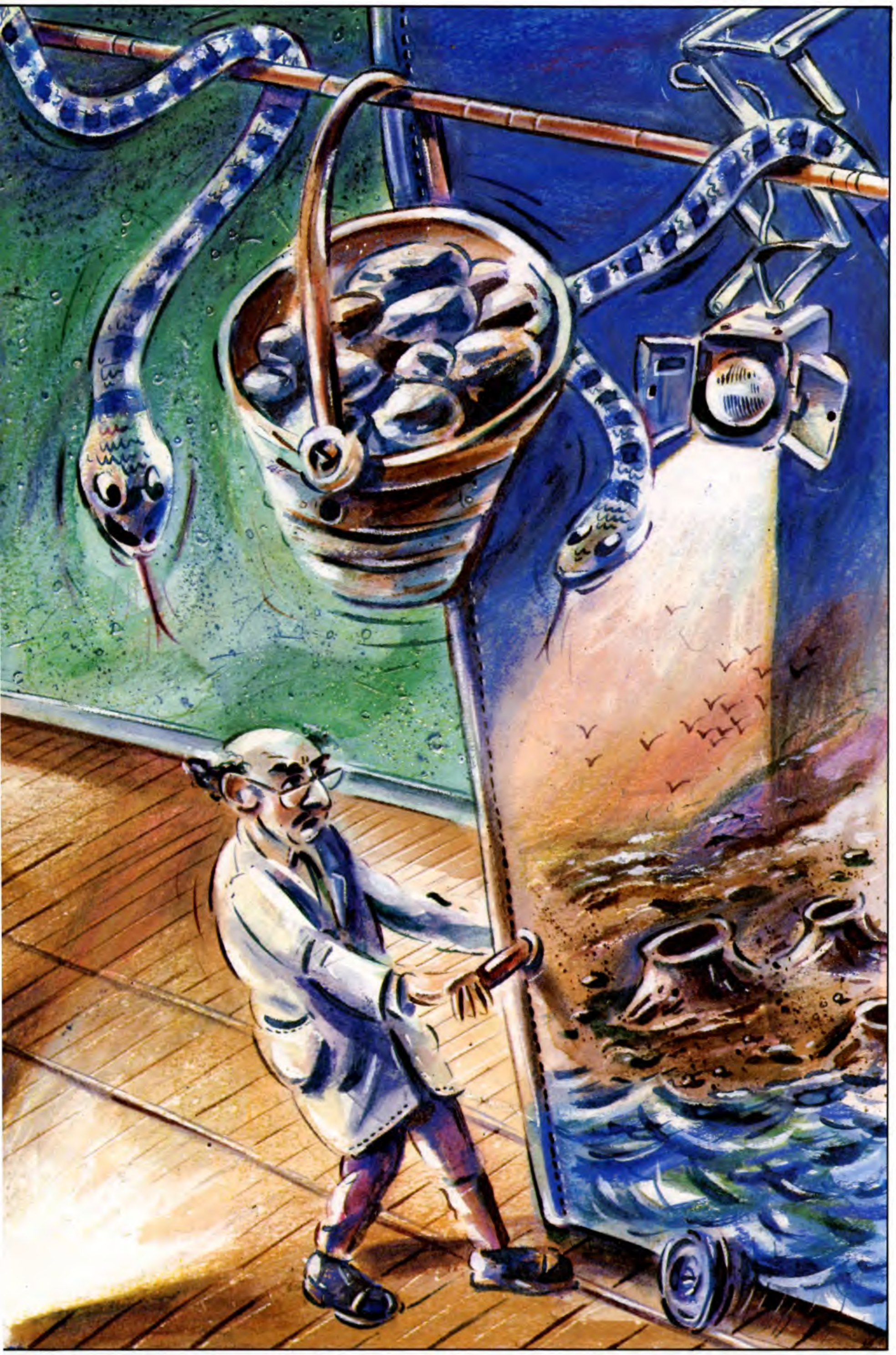
The **ld a,(ix+0)** instruction loads the accumulator with the last byte of the contour data. The zero offset is used here because the indirect addressing with the IX register has to be indexed. There is no **ld a,(ix)** instruction. The IX register is then decremented so that it points to the next byte of data.

The **cp 33** compares the contents of the accumulator with 33—the data byte that tells the routine that the slope goes down. If the contents of the accumulator are not 33—in



- PICKING UP THE PROFILE DATA
- SCROLLING IT ON
- THE EDGE GRAPHICS
- COLUMNS OF COLOUR
- REDEFINING CHARACTERS





other words the slope continues flat—**jr nz,lv** jumps straight on to the **lv** routine.

If it is going down the B counter and the Y coordinate of the landscape—stored in 57,328—are decremented. And the flag in 57,329 is set to 1. Then the program is ready to go into the **lv** routine.

## THE LV ROUTINE

The Y coordinate of the landscape is loaded into the B register. This has to be done via the accumulator as the B register cannot be addressed indirectly from a memory location, only from a register. There is no **ld b,(57328)** instruction.

HL is then loaded with 31, the screen position of the top right-hand corner of the screen. And the accumulator is loaded with 45, the number that will give cyan on cyan. The **lg** routine is then **call**ed. This prints a block of spaces from the HL position downwards, in the colour specified by A, B spaces long.

When it returns, **ld bc, 57264** loads BC with the position of the image data for sloping ground. The slope/flat flag in 57329 is then loaded into the accumulator and compared to the number one.

If the contents of the flag are not 1—that is they are Ø and the slope profile is flat—the **jr nz,mp** jumps straight to the **mp** routine. If the flag is set to 1 and so the slope is going down, **ld bc,(57329)** reloads BC with the address of the image data for sloping ground. You should be able to pick these two pieces of image data out of the graphics data supplied in the last part of Cliffhanger.

Whether the landscape is sloping or flat, the processor now enters the **mp** routine.

## THE MP ROUTINE

To print the character squares which mark the border between the sky and the land needs two colours. So **ld a,44** sets the ink colour to green. The background colour remains cyan. Then the **print** routine given in part one of Cliffhanger is called again which prints the top of the landscape.

The Y coordinate of the horizon is then loaded into the B register and it is subtracted from 23 to give the number of character

squares left below the horizon line. The result appears in A and is then transferred back into B where it is required when the **lg** routine is **call**ed. Note that the full 24-line screen including the two edit lines usually reserved by the system is being used.

The **ld a,32** sets the colour to green on green. Then 32 is loaded into DE and added to the print position in HL. This moves the print position one character square down the screen. The **lg** routine is then **call**ed to print the block under the horizon in green.

The **pop bc** pulls the column counter back off the stack. Then **djnz** decrements it and jumps back to the beginning of the whole procedure again to cope with the next column.

## SCORING

On the screen you also need an area set aside to print the score in. So HL is loaded with 49, to set the print position. The string length, 12, is loaded into B. And A is set to 41, to give blue on cyan.

IX is then loaded with 57973, which is the position of the memory byte being used to store the score. And the **me** routine given in part one of Cliffhanger is **call**ed again. This translates the score into ASCII characters and prints them on the screen.

After that the **elb** routine is **call**ed. This is the **e**xtra **l**evel **b**its routine which adds the snakes and the pits when you move onto a higher level. The routine hasn't been written yet and you will see that the **elb** label marks a single **ret** at the end of this routine. So for now the processor will return straight away. This **ret** will be overwritten later though when the proper **elb** routine is given.

When the processor returns from the **elb** routine it hits another **ret**. At the moment this returns the processor to BASIC. But when the whole program is finished it will return to the main driver program which will call the next routine in the game.

## JUST SCROLLING

The label **scl** marks the beginning of the scroll routine. This scrolls the instruction page off and the slope on sideways.

HL is loaded with 16,384, the start of the display file. B is loaded with the number of rows in the display file and attribute file. And C is loaded with 31, the number of columns per row.

The display file pointer in HL is then incremented. So the **ld a,(hl)** loads the contents of the second screen location into the accumulator. HL is then decremented so it points to the first screen location and the contents of the A are loaded back there.

The HL pointer is then incremented again and the column pointer in C is decremented. And if this hasn't been decremented to zero at the end of the screen, **jr nz,lpj** loops back to deal with the next column. But if the end of the screen has been reached the pointer in HL is incremented again.

The **djnz** instruction always acts on the BC register pair. And as C is always zero at this point, it effectively decrements B and loops back to the beginning of the routine to scroll the next row. When it has scrolled the last row, B has counted down to zero and the **ret** returns the processor to the place where the scroll routine was **call**ed.

## THE LG ROUTINE

The **push bc** stores the counter in BC on the stack. BC is then loaded up with 15,616. This is the address of the beginning of the Spectrum's character set—and it starts with a space. So when the **print** routine is **call**ed again, it prints a space of the appropriate colour on the screen. 32 is loaded into DE and added to HL to move the print position down one l character square.

The **pop bc** gets the counter back again and **djnz lg** decrements it and jumps back to print the next character square down the screen, unless the last character square has been dealt with. In that case the processor moves onto the **ret** and returns to the place the **lg** routine was **call**ed from.

## [Commodore]

Unfortunately, it is very difficult to scroll on the scenery on the Commodore. It is easy to make it scroll between scenes once the first one has been put up—as all the scenes are basically the same it is just a matter of taking what comes off one end of the screen and putting it back on at the other. But putting a fresh bit of scenery on the screen is a different matter. So it is going to be dropped like a flat as soon as the instruction page has disappeared.

But first you need to paint the flat. The following BASIC program carries the cliff data. It puts it up on the screen so that you can see the outline of the scene. It won't make much sense at the moment because it is put up as ASCII strings. But you will be able to see what goes where.

The BASIC program then stores the screen in another area of memory where the machine code can pick it up. When it does, it will convert it into ROM graphics so what is shown on the screen then actually looks like a cliff!

```
10 PRINT"○◙◙◙◙◙◙◙◙◙
   ◙◙◙◙◙◙◙◙□££○]££○]
   ££○]££££○]£○]£££○]£££○]£○
   ]£○]££";
20 PRINT"○]£○]££££○]£"
30 FORX=ØTO39:F=Ø:FORY=ØTO24
40 SC=1Ø24:CL=55296
50 P=INT(RND(1)*3)+1:IFP=1THENV=3Ø
60 IFP=2THENV=31
70 IFP=3THENV=63
80 C=X+Y*4Ø:IFF=1THENPOKESC+C,V:
   POKECL+C,Ø
90 IFPEEK(SC+C)=28ORPEEK(SC+C)=29
   THENF=1
100 NEXTY,X
110 PRINT"☰◙LIVES◙▮▮▮▮▮▮▮▮▮▮
   LEVEL◙▮▮▮▮▮▮▮▮▮▮SCORE□ØØØØØ
   Ø☰"TAB(27)"◙▙▙▙▙="
120 FORI=ØTO999:P=PEEK(SC+I):POKE
   13312+I,P:NEXT
130 PRINT"☰FINISHED":STOP
```

You can SAVE the data by using your machine code monitor. The data table runs from 13,312 for 999 bytes.

## PAINTING THE SCENERY

The following routine picks up the scenery data and puts it on the screen as part of the game:

```
ORG 25344           INC $FB
LDA #$ØØ            BNE $631F
STA $FB             INC $FC
LDA #$Ø4            INC $FD
STA $FC             BNE $6325
LDA #$34            INC $FE
STA $FE             LDA $FB
LDA #$ØØ            CMP #$E8
STA $FD             BNE $6312
LDY #$ØØ            LDA $FC
LDA ($FD),Y         CMP #$Ø7
STA ($FB),Y         BNE $6312
JSR $515Ø           RTS
```

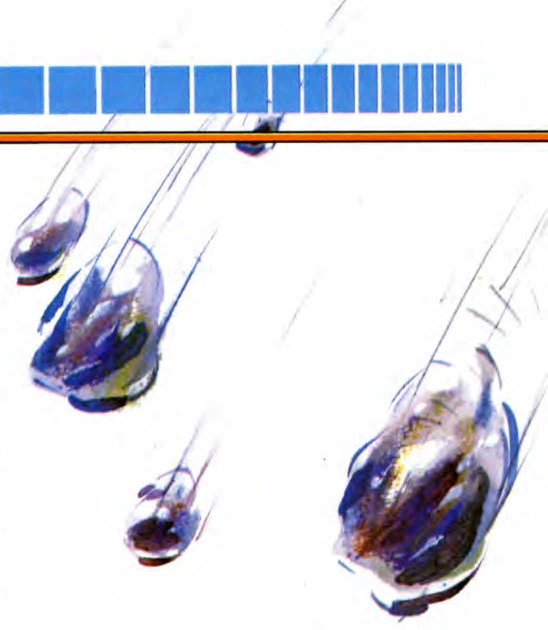
You can assemble and SAVE this program now, but do not call it. It calls the subroutine given below and without that in memory it will crash.

## PAINTING BY NUMBERS

The first four instructions set the screen pointer. And the second four establish another zero-page pointer for the data.

Then the Y offset is set to zero because you're going to increment the zero-page. And the byte pointed by the data pointer is loaded into the accumulator and stored in the screen position pointed. The routine at $515Ø is then called. This is given below. What it does is set the colours.

The next six instructions increment the pointers. The BNE instructions in that part of



the routine check whether the low byte of the pointer has reached the end of page and jumps the subsequent INCs—which increment the high byte—if the end of the page has not been reached.

The six instructions after that check to see if the end of the screen has been reached. And exits the program if it has been. The RTS here will be overwritten when the rest of the program is added.

## ADDING COLOUR

The following routine fills in the colour:

```
ORG  2Ø816
LDY  #$ØØ
LDA  ($FB),Y
STA  $Ø384
LDA  $FC
CLC
ADC  #$D4
STA  $FC
LDA  $Ø384
CMP  #$1C
BEQ  $5181
CMP  #$1Ø
BEQ  $5181
CMP  #$3F
BEQ  $5186
CMP  #$1E
BEQ  $5186
CMP  #$1F
BEQ  $5186
LDA  #$Ø2
STA  ($FB),Y
LDA  $FC
SEC
SBC  #$D4
STA  $FC
RTS
LDA  #$Ø5
JMP  $5177
LDA  #$Ø1
JMP  $5177
```

The subroutine starts off by loading up the same byte of data that the main program has been dealing with and stores it in $Ø384. This is a temporary store because the accumulator has to be used for something else just for the moment.

The next four instructions add $D4 to the high byte of the screen pointer in $FC. This shifts the pointer from its position in the graphics screen—which starts at $Ø4ØØ—to the corresponding position on the colour screen—which starts at $D8ØØ.

Then the data byte is loaded back from $Ø384 into the accumulator. Then it checks for various control codes in the data. Depending on the control code found, the processor branches to the instruction at $5181 or the one at $5186.

The instruction at $5181 is LDA #$Ø5 which loads the accumulator with the ink colour green. And LDA #$Ø1 at $5186 loads up ink colour white. This gives the green of the grass and the white is the white of the cliff itself. The two tones of the grass and the cliff are given by graphics characters which mix the ink colour with the paper.

If none of the control characters are picked up, the accumulator is loaded with red. So everything that is on the screen, which is not green or white is coloured red.

No, this doesn't mean the sky is red. Willie is not a shepherd. Red is the ink colour so it only appears when data is written on the screen. So the number of lives, level and score are written in red, but the sky is in the paper colour, grey. Unfortunately, it was rather overclouded on the day Willie decided to have his picnic—in the Commodore version at least.

If the character is printed in white or green, the processor jumps back to the instruction at $5177, which is the one after the red colour is set for the rest of the data. This stores the chosen colour on the colour screen in the appropriate place.

The next four instructions subtract $D4 from the high byte pointer in $FC, to move it back from the colour screen to the graphics screen. So when the RTS returns to the main routine the next graphics character can be picked up and put on the screen.

Rather a lot of BBC programming has to be given at this point. You need a couple of routines to deal with user-defined graphics and a third to define the colours. Don't forget to key in PAGE=&3ØØØ and NEW before you key in this program.

```
3Ø FORPASS=ØTO3STEP3
4Ø P%=&17D4
5Ø [OPTPASS
6Ø .Chardef
7Ø LDA#23
8Ø JSR&FFEE
9Ø TXA
1ØØ JSR&FFEE
11Ø LDA#Ø
12Ø STA&71
13Ø TYA
14Ø ASLA
15Ø ROL&71
16Ø ASLA
17Ø ROL&71
18Ø ASLA
19Ø ROL&71
2ØØ CLC
21Ø ADC#&34
22Ø STA&7Ø
23Ø LDA&71
24Ø ADC#&15
25Ø STA&71
26Ø LDY#Ø
27Ø .Lb1
28Ø LDA(&7Ø) ,Y
29Ø JSR&FFEE
3ØØ INY
31Ø CPY#8
32Ø BNELb1
33Ø RTS
34Ø .Pt
35Ø STA&72
36Ø TXA
37Ø PHA
38Ø TYA
39Ø PHA
4ØØ LDA&72
41Ø AND#&8Ø
42Ø BNELb2
43Ø LDA&72
44Ø JSR&FFEE
45Ø JMPLb4
46Ø .Lb2
47Ø LDA&72
48Ø AND#&7F
49Ø TAY
5ØØ LDX#&FF
51Ø JSRChardef
52Ø LDA#&FF
53Ø JSR&FFEE
54Ø .Lb4
55Ø PLA
56Ø TAY
57Ø PLA
58Ø TAX
59Ø LDA&72
6ØØ RTS
61Ø .Def
62Ø LDX#Ø
63Ø STX&72
64Ø .Lb3
65Ø LDA&72
66Ø CLC
67Ø ADC#224
68Ø TAX
69Ø LDY&72
7ØØ JSRChardef
71Ø INC&72
72Ø LDX&72
73Ø CPX#23
74Ø BNELb3
75Ø RTS
76Ø ]NEXT
77Ø DATA6,1,5,Ø,3,
   7,4,6,2,1,5,Ø,3,7,6,6
78Ø FORA%=&1845
   TO&1854:READ?A%:
   NEXT
79Ø FORPASS
   =ØTO3STEP3
8ØØ P%=&1855
81Ø [OPTPASS
82Ø .Colour
83Ø LDX#Ø
84Ø .Lb5
85Ø LDA#19
86Ø JSR&FFEE
87Ø TXA
88Ø JSR&FFEE
89Ø LDA&1845,X
9ØØ JSR&FFEE
91Ø LDA#Ø
92Ø JSR&FFEE
93Ø JSR&FFEE
94Ø JSR&FFEE
95Ø INX
96Ø CPX#16
97Ø BNELb5
98Ø RTS
99Ø ]NEXT
```

When you have keyed in this program SAVE it, then RUN it. To test it, the rest of the program must be in memory then key in the following instructions:

```
PAGE=&2ØØØ
NEW
MODE 2
CALL &182D
FOR A%=224 TO 255:VDU A%:NEXT
```

This tests the first routine. To test the second key in:

```
FOR A%=128 TO 211:CALL &18Ø3:NEXT
```

And to test the third routine:

```
CALL &1855
```

## USER DEFINED GRAPHICS

The first routine redefines some of the character set as user-defined graphics. As always when you are dealing with the screen, the routine at &FFEE is called and directed by

the parameter in A. A value of 23 in A tells the routine that you want to redefine a character.

The parameter in X when you enter this program is the ASCII code of the character in the machine's character set which you want to redefine. And the parameter in Y is the number of the user-defined graphic you want it redefined as.

So the contents of X are transferred into A and the &FFEE routine is called again. This tells the machine which character you want to redefine.

The new data for the user-defined graphic is stored in a data table. Characters take up an eight by eight square. So the data for each character takes up eight bytes—which are each eight bits long. So to count along the data table, you have to multiply the new character number by eight. The result is going to be stored in &7Ø and &71.

The number in the Y register must be less than 255—that's the capacity of an eight-bit register. So the high byte of the answer is set to zero before you start.

The contents of the Y register is than transferred into the accumulator where it can be manipulated. The contents of the accumulator is then shifted to the left and the contents of &71, the high byte of the answer store, are than rotated to the left.

## SHIFTS AND ROTATES

When performing a multiplication on a number which might yield a two byte result, the different properties of a shift and a rotate become very useful.

A shift left moves all the bits one place to the left, effectively multiplying the contents by two. Bit zero is filled with Ø and the overflow from bit seven goes into the carry flag.

A rotate left also shifts all the bits one place to the left. But it loads bit zero with the contents of the carry flag and rests the carry flag with the overflow from bit seven. In other words, it shuffles—or rotates—all the bits round, rather than just shifting them along.

Using the two of them in conjunction, as here, effectively gives a 16-bit shift. If there is any overflow from the ASL instruction it is automatiacally picked up by the ROL through the carry flag.

Here the ASL and ROL combination is used three times, multiplying the Y parameter by eight. Then &34 is added to the low byte and &15 is added to the high byte. The data table starts at &1534.

## ENTER THE DATA

The Y register is then loaded with zero, then the accumulator is loaded indirectly from the location in the data table pointed to by &7Ø and &71, offset by Y. The &FFEE routine is called yet again and the first byte of UDG data is entered.

Y is then incremented, compared to 8 and the BNE Lb1 branches back to enter the next byte of the data table if Y hasn't clocked up to eight yet.

When it has clocked up that far, the processor hits the RTS and exits the routine.

You will notice that this routine redefines a character and is followed by nine parameters. The first is the number of the character to be redefined and the next eight are the data for the new character. This is exactly what is fed into &FFEE subroutine and you'll find the 23 in line 7Ø.

## WHICH CHARACTER?

The next little routine decides whether a ASCII character or a UDG is to be printed. The character codes up to 127—the alphabet, the numbers and the punctuation marks—are going to be printed as normal but the codes



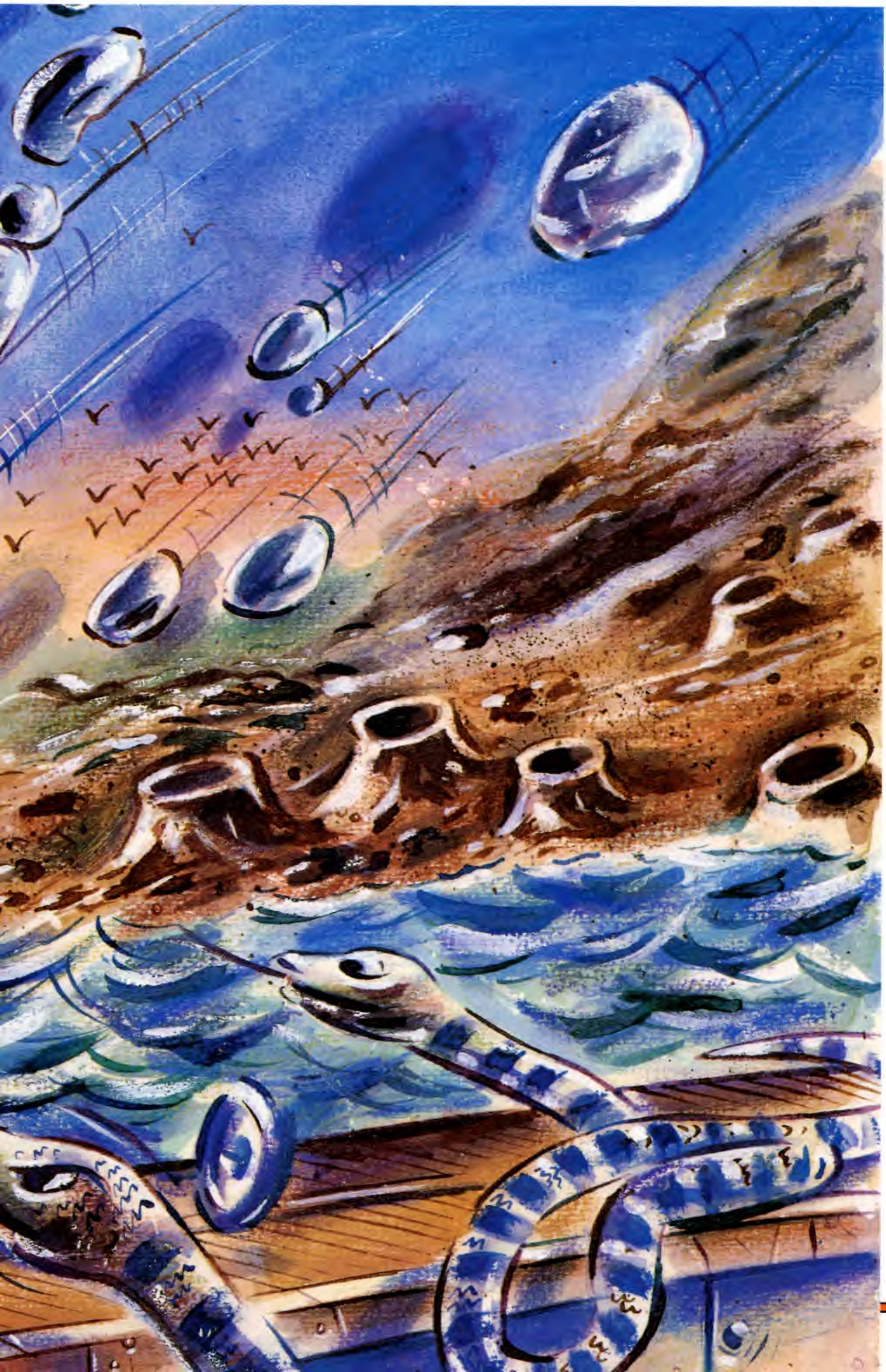
from 128 to 255 are going to be UDGs.

The first thing that has to be done is to store the contents of A, X and Y. They may be required later. A is stored in &72, then X is transfered into A and and pushed onto the stack and the Y register is transferred into A and pushed onto the stack.

Then the contents of &72 is loaded back into the accumulator and ANDed with &8Ø. This checks to see if the most significant bit is set—in other words, if the number in the accumulator is greater than or equal to 128.

The AND instruction sets the zero flag if the result of the AND is zero. So if the number in A is 127 or less, the BNE instruction does not branch and the processor continues. The accumulator is again loaded with the contents of &72 and that corresponding ASCII character is output to the screen. Then the processor jumps on to the routine that restores the contents of A, X and Y.

But if the number in the accumulator is 128 or more, the BNE instruction branches the processor onto the label Lb2, where the accumulator is loaded up with the contents of &72 yet again and ANDed with &7F. This resets the most significant bit to Ø and leaves the rest of the bits alone, effectively subtracting 128 from the ASCII to give the number of the UDG required. This number is then transfered into the Y register.



X is then loaded with 255 and the Chardef routine above is called. This redefines character 255 as the one you specify in Y. Then 255 is loaded into the accumulator and the &FFEE routine is called. This prints the redefined character 255 on the screen.

The rest of the routine, from Line 55Ø to 59Ø, restores the contents of A, by reloading it from &72 yet again, and X and Y by pulling them off the stack.

## STOCK CHARACTERS

Some UDG characters are going to be used frequently during the program and you don't want to have to define them every time they are used. It would be much more convenient if you could just go straight to central casting and pull them out.

The routine carried in Lines 62Ø to 75Ø redefines characters 224 to 246 as UDGs Ø to 22. This leaves these characters permanently changed for the duration of the game and they can be printed up immediately at any time by loading the accumulator with one of the appropriate numbers and calling the routine at &FFEE without having to go through the whole rigmarole of redefining a character each time.

The routine is initialized by loading X with zero and storing that in the zero page location &72. Once past the label Lb3 which marks the beginning of a loop, the contents of &72 are loaded back into the accumulator. Then 224 is added to take it to the start of the characters to be redefined.

The result of the additon is transferred into the X register. Y is loaded with the contents of &72 and the Chardef routine is called. This redefines the character given by 224 plus the number of the loop you're onto as the character pointed to by the number of the loop.

The contents of &72 are then incremented to move onto the next character. X is loaded with the result so that it can be compared to 23. The processor branches back and redefines the next character until it has gone round the loop 23. Then it drops out and returns.

## COLOURING

Next the colour has to be defined. The LDA # 19 and JSR&FFEE in Lines 85Ø and 86Ø acts like a VDU19, which changes the original colour.

The value in X is then transferred into the accumulator. This is the number of the loop you're on and, when &FFEE is called, is the number of the colour that is to be defined.

The new colour number is supplied by the data in Line 770 and is picked up by the LDA&1845,X in Line 890. Then &FFEE is called. The VDU 19 instruction always ends with three zeros which are left open for future expansion. This is done by loading A with 0 and calling &FFEE three times.

X is incremented to move onto the next colour and the next byte in the data table and compared to 16 to see whether all 16 colours have been defined. When they have the processor leaves the loop and exits.

## DISPLAY AND PLAY

This program prints up the first screen.

```
80 DATA 17,8,31,19,6
90 FORA%=&1877TO&187B:READ?A%:NEXT
140 DATA44,61,24,61,21,61,24,43,25,61,21,
   47,13,51,12,51,14,54
150 FORA%=&187CTO&188D:READA$:
   ?A%=EVAL("&"+A$):NEXT
200 DATA239,239,0,239,241,240
210 FORA%=&188ETO&1893:READ?A%:
   NEXT
250 FORPASS=0TO3STEP3
260 P%=&1894
270 [OPTPASS
280 .Screen
290 LDX#0
300 .Lb1
310 LDA&1877,X
320 JSR&FFEE
330 INX
340 CPX#5
350 BNELb1
360 LDX#0
370 STX&70
380 .Lb2
390 LDA&187C,X
400 LSRA
410 LSRA
420 LSRA
430 LSRA
450 PHA
460 TAY
470 LDA&188D,Y
480 STA&71
490 LDA&187C,X
500 AND#&F
510 TAX
520 .Lb3
530 LDA&71
540 JSR&FFEE
550 LDA#8
560 JSR&FFEE
570 TXA
580 PHA
590 LDA#&86
600 JSR&FFF4
610 STY&72
620 LDA#9
630 JSR&FFEE
640 .Lb7
650 LDA#8
660 JSR&FFEE
670 LDA#10
680 JSR&FFEE
690 LDA#224
700 JSR&FFEE
710 INY
720 CPY#30
730 BNELb7
740 LDA#31
750 JSR&FFEE
760 TXA
770 JSR&FFEE
780 LDA&72
790 JSR&FFEE
800 PLA
810 TAX
820 PLA
830 PHA
840 AND#1
850 BEQLb4
860 LDA#9
870 JSR&FFEE
880 .Lb4
890 PLA
900 PHA
910 AND#2
920 BEQLb5
930 LDA#8
940 JSR&FFEE
950 .Lb5
960 PLA
970 PHA
980 AND#4
990 BEQLb6
1000 LDA#10
1010 JSR&FFEE
1020 .Lb6
1030 DEX
1040 BNE Lb3
1050 PLA
1060 INC&70
1070 LDX&70
1080 CPX#18
1090 BNELb2
1100 LDA#241
1110JSR&FFEE
1120 RTS
1130 ]:NEXT
```

After you have SAVEd and RUN this program key in:

```
PAGE=& 2000
NEW
MODE 2
CALL 6292
```

The result of CALLING this program will look strange unless you have the graphics data in memory at the same time. The colours are going to look funny anyway—the program that redefines them is going to be CALLED later.

## WHAT'S THE DATA?

Lines 80 and 90 READ in the DATA which selects colour 8 and moves the cursor to its start position at 19,6. The DATA in Line 140 supplies details of the slope encoded bit by bit. Bit seven is not used. Bits six to four specify which way the slope is going next. Bit six set to 1 means it stays level. Bit five set to 1 means that it is going left and bit four set 1 means that it is going right. On the Acorn computer's screen the slope is doubled back on itself to give Willie enough height to scale.

The last three bits stand for the number of character squares in the direction specified.

The DATA in Line 200 is the character data. These numbers define the shape of the top of the slope on the screen.

## SLOPING OFF

The routine in Lines 290 to 350 set the colour and position the cursor. The LDA&1877,X picks up the DATA given in Line 80 which is then output through the screen routine by the JSR&FFEE. When the first data byte 17 is output it gives a VDU 17, so the colour is defined as the following byte which is 8. And colour 8 has been redefined in the routine above to COLOUR 2 which is green. The background colour stays as it was.

The 31 in the DATA gives a VDU 31, which positions the cursor at the point specified by the two bytes that follow.

The main routine starts with Line 360. Line 390 reads in the display data. To isolate the direction data in bits six to four, four logical shifts right are down. This shifts bit four into bit zero, bit five into bit one, bit six into bit two and shoves bits zero to three out of the register. Line 450 saves the direction data by pushing it onto the stack.

Line 460 transfers the same direction data into the Y register so that it can be used for indexing. And Line 470 loads up the byte of character data. This is stored in &71.

The display data is loaded up again in Line 490 and it's ANDed with F to isolate bits zero to three. The result is then transferred into X so that it can be used as an index.

The character byte just stored in &71 is output to the screen by Lines 530 and 540. Lines 550 and 560 then load up 8 and output that to the screen. This moves the cursor back to the position it has just printed in.

The index is transferred into A and stored on the stack. Then A is loaded with &86 and the routine at &FFF4 is called. This reads the position of the cursor and returns the X and Y values in the appropriate registers. The Y values is stored in &72.

The cursor is then moved forward again by loading 9 into A and calling &FFEE. This may seem a little unnecessary as the Y coordinate is the same in the next position along the screen. But the print position may have been at the end of the screen and cursor would have moved down a line.

Then, lo and behold, in Lines 650 and 660 the cursor is moved back again! But this is inside the loop that prints the green spaces under the slope, so in this loop the cursor has to be shifted back to the same position—and then moved down one line—at the beginning.

The move down one line is done by loading A with 10 and calling &FFEE. Then character 224 is printed on the screen. This has been redefined as a solid block. Y is then incremented and the loop is executed again until it reaches 30 which means the cursor has reached the bottom of the screen.

Lines 750 to 790 return the cursor to its original position. The X index is then pulled off the stack again and transferred back into the X register. The direction data byte used to draw the top of the slope is then pulled off the stack and pushed back on again. This copies it back into A and leaves it on the stack.

## WHICH WAY NOW?

A series of ANDs look at the state of the direction data and decide which way the slope is going next. AND#1 looks at bit zero and BEQ branches if it is not set.

If it is set, the slope is continuing right and

the branch is not made. A is closed with 9 and the screen routine is called. This moves the cursor to the right.

Lines 890 and 900 copy the direction data back into A and it is ANDed with 2 to check whether bit one is set. If it's not, the processor branches forward. If it is, the slope is going to the left. So A is loaded with 8 and the screen routine is called. This moves the cursor to the left.

Lines 960 and 970 copy the direction data back into A again, then AND #4 checks to see if bit three is set. If it's set, A is loaded with 10 and the screen routine is called again. This moves the cursor down.

The counter in X, which is the number of spaces that the slope continues in any particular direction is decremented, and the processor loops back if it hasn't counted down to zero.

If it has, that particular section of the slope is finished and the processor moves on. The main loop counter in &70 is then incremented. This was intialized to zero at the start of the main loop in Lines 360 and 370. Its contents are now loaded up into the X register and compared to 18—there are 18 distinct sections in the slope. If the counter has not counted up that far, the processor branches back and starts on the next section of the slope. If it has, the processor continues.

A is loaded with 241 and the screen routine prints character 241 on the screen. This is the last character of the slope and is one of the UDGs you redefined earlier.

The following program scrolls on the dragon's scenery. This is a bit different from that on the other machines because of the limitations of the Dragon and Tandy's colour set. With green grass on the slope and a blue sea, you have little alternative but to have a yellow sky! But then it is a very hot day.

```
        ORG     19109
        JSR     MODE
        JSR     GCLS
        LDX     #5631
        LDY     #17503
        LDB     #32
LOOP    PSHS    B
        JSR     SCROLL
        JSR     PRINT
        PULS    B
        DECB
        BNE     LOOP
        LDY     #17604
        LDX     #1569
        JSR     PRSUN
        RTS
MODE    EQU     19182
GCLS    EQU     19161
```



```
SCROLL  EQU    19197
PRINT   EQU    19218
PRSUN   EQU    19267
```

Key this in, assemble it and SAVE it. But don't EXECute it at the moment. It won't work until the subroutines given below are in memory as well.

## MOVING THE MOUNTAIN

The first thing that has to be done is to put the computer into graphics mode and select the appropriate colour set. This is done by the subroutine MODE—which is called by JSR MODE. The screen is then cleared by jumping to the GCLS routine which sets it all to yellow.

LDX #5631 loads the X register with the left-hand end of the horizon which is at the right-hand end of the screen before it is scrolled on. LDY #17503 loads the Y register with the address of the start of the slope profile data. And LDB #32 loads B with 32, the number of columns on the screen. The column counter is then pushed onto the stack for safekeeping.

Then the SCROLL routine is called, which scrolls on the first column of the scenery. After that the PRINT routine is called. This prints the column of green below the horizon.

The column counter is then pulled off the stack again and decremented. Then the processor loops back to deal with the next column unless, of course, the last column has been completed.

If it has the processor continues and loads Y with the start of the data for the sun. It loads X with the position of the sun, and then jumps to the PRSUN subroutine.

## CURIOUS YELLOW

The GCLS routine clears the screen by turning it all yellow—the sky colour in this version of the game.

```
        ORG    19161
GCLS    LDX    #1536
        LDA    #85
GCLSI   STA    ,X+
        CMPX   #7680
        BLO    GCLSI
        RTS
```

The X register is loaded with the address of the beginning of the screen. A is loaded with 85, which is the code for the colour yellow.

STA ,X+ stores the yellow in the position pointed to by X and increments X. X is then compared to 7,680 the first location past the end of the screen and the BLO GCLS branches back to fill the next character square with yellow if the end of the screen has not yet been reached. When it has the routine returns to where it was called from.

## A LA MODE

The following four subroutines can be entered together as they follow on from the previous one.

To change graphics mode you have to address the Video Display Generator chip and the Synchronous Address Multiplexor chip. These have to be set up for the new graphics configuration you require.

A is loaded with the number 229 which is stored in memory location FF22. This memory location controls the control lines for the VDG and other output functions. Each bit of this byte controls a separate function.

Here the number 229—11100101 in binary—sets the control lines. The 1 in bit seven sets the VDG to graphics mode, as against alphanumeric. Bits six and five are set to give graphics mode P3. Bit three switches between colour sets—0 here gives colour set one which comprises green, yellow, blue and red.

Bits two, one and zero control have nothing to do with the VDG chip. They control the RAM size, single-bit sound and the printer respectively and are usually set to 101. So when you change the setting of the control lines, make sure you put 101 back in these bits—unless, of course, you have some good reason for changing them.

When you change the settings of the control lines of the VDG chips you have to change the control register of the SAM chip as well. The SAM chip has a 16-bit register whose bits correspond to memory locations FFC0 to FFDF. You'll notice that there are 32 memory locations in that range. Each bit of the control register is set by writing to the odd-numbered byte associated with it, and cleared by writing to the even-numbered byte. And when you write to these bytes you should put into them the same values you put into the VDG control location. The bits that are set here tell the SAM chip that the screen starts at 1,536.

```
        ORG    19182
MODE    LDA    #229
        STA    65314
        STA    65475
        STA    65477
        STA    65479
        RTS
```

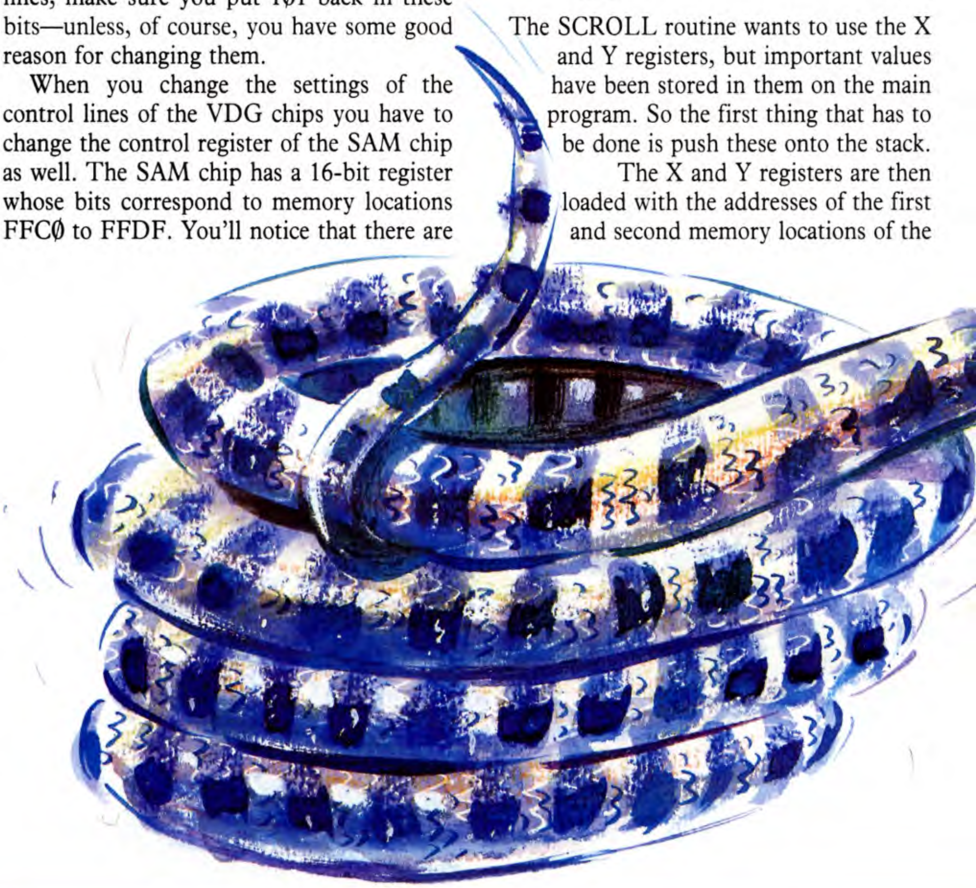
## SCROLL ON

The SCROLL routine moves the scenery on from the right.

```
SCROLL  PSHS X,Y
        LDX    #1536
        LDY    #1537
SCRO    LDA    ,Y+
        STA    ,X+
        CMPX   #7679
        BLO    SCRO
        PULS   Y,X
        RTS
```

The SCROLL routine wants to use the X and Y registers, but important values have been stored in them on the main program. So the first thing that has to be done is push these onto the stack.

The X and Y registers are then loaded with the addresses of the first and second memory locations of the



screen. The contents of the second screen location is then loaded into first. Both address pointers are updated. Then the value of X is compared with the address of the one before the last screen location and the processor branches back to shift the contents of the third screen location into the second, and so on, if the last location has not been shifted.

You will notice that this not only scrolls everything on the screen one location to the left, it also brings the contents of the last screen location on the left round into the last screen location on the right, one line above. This does not matter as the last column is going to be overwritten with the new bit of scenery that is about to appear.

And when the contents of the last memory location on the screen has been moved into the location before last, the contents of the X and Y registers which were stored on the stack at the beginning of the routine are pulled off again. Then the processor returns to the point where the SCROLL routine was called.

## PRINTING THE NEW SCENERY

The extreme right-hand column of the screen has to be dealt with separately. The new scenery is printed in there by this routine:

```
PRINT   PSHS    X
        LDA     ,Y+
        SUBA    #33
        BNE     PRZ
        PULS    X
        LEAX    -256,X
        PSHS    X
        PSHS    Y
        LDY     #17536
        LDB     #8
PRI     LDA     ,Y+
        STA     ,X
        LEAX    32,X
        DECB
        BNE     PRI
        PULS    Y
PRZ     CLR     ,X
        LEAX    32,X
        CMPX    #7680
        BLO     PRZ
        PULS    X
        RTS
```



This time the data pointer held in the Y register is going to be needed. And the horizon height held in the X register might have to be adjusted too—if the routine is not on a flat bit of the cliff. But for now X has to be stored on the stack.

The byte of data pointed to by the pointer in Y is loaded into A and the pointer is incremented. Then 33 is subtracted from it. A 33 in the data—which is the ASCII for !—means that the slope continues flat. In the programming it means that the BNE instruction following makes the processor branch to PRZ label. If not and the slope is set to rise, the processor continues.

The horizon height is then pulled off the stack and 256 is taken away from it. 256 is 32 × 8, so the X pointer is moved up the screen eight pixel lines or one character square. This is stored back on the stack as it will be needed when the next column has to be dealt with. The data pointer is pushed onto the stack to preserve it too.

The Y register is then loaded with 17,536 which is the start of the data for a sloping piece of horizon. B is loaded with eight—it is going to be used as a counter to count the eight bytes of data that are needed to make up the sloping section of horizon.

LDA ,Y+ loads A with the first byte of data and increments the pointer. This is stored in the pixel position pointed to by the address in X. In this mode the pixels are set two at a time, by two bits of data. With two bits, there are four possible values—one for each of the four colours in the colour set. How these are set is covered in the article on Better Graphics on pages 248 and 249.

B is decremented and the processor branches back to pick up the next byte of data and fill in the next line of pixels, unless B has been counted down to zero and the last line has been dealt with.

When B reaches zero, the slope data pointer is pulled back off the stack and the processor goes into the PRI routine. This is the same routine it would have jumped to earlier, if the slope had continued flat. It fills in the solid blocks of green that form the land.

## GOING GREEN

CLR ,X clears the contents of the address pointed to by X. X, you remember, points to the screen position you're dealing with and clearing it—setting the contents to 0—fills it with the colour green. LEAX 32,X adds 32 to the value of X and moves the pointer one position down the screen.

CMPX #7680 checks to see whether the processor has reached the end of the screen. If it hasn't, it branches back to next pixel line with green. If it has, the horizon height is pulled off the stack again—whether it has been updated or not—and returns to the place it was called from.

## HERE COMES THE SUN

The print and data positions for the sun has been given before the PRSUN routine is given and the following routine fills in a patch 32 by 30 in the sky:

```
PRSUN   LDB     #30
PRSUNI  PSHS    B
        LDB     #4
PRSUNZ  LDA     ,Y+
        STA     ,X+
        DECB
        BNE     PRSUNZ
        LEAX    28,X
        PULS    B
        DECB
        BNE     PRSUNI
        RTS
```

B is loaded with 30 to count down the lines of pixels to be filled. This counter is then pushed onto the stack and B is loaded again with 4. This gives a patch four bytes—or 4 × 8, 32 bits—wide on the screen. If you want to fill in a graphic pattern 30 by 30 pixels, you would have to put it on a patch 30 by 32 pixels—you would have to use four bytes and leave two columns of pixels in the background colour.

Again, LDA ,Y+ picks up the appropriate piece of data and increments the pointer and STA ,X+ stores it in the appropriate screen position and increments that. B is decremented, counting across the four horizontal screen locations.

When the last of the row has been filled, LEAX 28,X adds 28 to X, moving the pointer onto the first location of the next row. The vertical counter is pulled back off the stack, decremented and—if the last row hasn't been dealt with—BNE PRSUNI branches back to deal with the next row.

If the last row has been dealt with the RTS returns the processor to the place in the main program it was called from.

# SENDING SECRET MESSAGES-2

**Take a look at some more subtle ways of concealing missives that might fall into the wrong hands. But first, find out the code-breaking methods that might be employed**

As the article on pages 960 to 965 showed, there are many different approaches to the problem of coding sensitive information. Some of these are relatively easy to crack, but with the aid of computers, it has been possible to employ even more complicated methods.

## CODE BREAKING

As the cryptographers struggle to invent better and more secure codes, so the code-breakers strive to frustrate them. A powerful tool when trying to decode the simpler transpositions or substitution ciphers is the letter frequency count. In English, the letters E, T, A, O, N, I, S—in that order—occur most frequently. So, if in the encoded text the letters ETAONIS still appear most frequently, the odds are that you will be dealing with a transposition code. Should other letters occur more frequently, then you must consider the possibility of a substitution cipher.

In either event it will be necessary to count the letter frequencies in the message. This is both time-consuming and susceptible to error. The frequency distribution program given below comes in useful here. All you need to do is type in your text and when you've finished, the computer displays the complete letter frequency count.

The picture below shows the letter distribution for 100 words from a newspaper article. You will see that the numbers are in good agreement with the ETOANIS principle. Now type in the program which shows how this works in practice:




**The table of letter frequency is useful for code-breaking**

```
15 POKE 23658,8
20 BORDER Ø: PAPER Ø: INK 7: CLS
30 PRINT TAB 8;"FREQUENCY COUNT"''
40 PRINT TAB 12;"WARNING"''
50 PRINT FLASH 1;AT 4,6;"DO NOT LEAVE
   SPACES";AT 5,9;"BETWEEN WORDS"''
60 DIM n(28)
70 PRINT "To end input of text and
   output□□□□□□□□results – type
   *"
80 FOR t=1 TO 28: LET n(t)=Ø:
   NEXT t
90 INPUT "Enter text ";a$
100 IF a$="*" THEN GOTO 180
110 CLS
120 FOR i=1 TO LEN a$
130 FOR j=1 TO 26
140 IF j=CODE (a$(i TO i))–64 THEN LET
   n(j)=n(j)+1
150 NEXT j
160 NEXT i
170 GOTO 90
180 CLS
190 PRINT "Letter□□□Freq'
   □□□Letter□□Freq"
200 FOR i=1 TO 13
210 PRINT TAB 2;CHR$ (64+i);TAB
   10;n(i);TAB 19;CHR$ (77+i);TAB
   27;n(13+i)
220 NEXT i
230 STOP
```

```
30 PRINT "♥▪>▪▪FREQUENCY
   COUNT"
50 PRINT "▪▪DON'T LEAVE
   SPACES":PRINT "BETWEEN
   WORDS !"
60 DIM N(28)
70 PRINT "▪TO END INPUT/OUTPUT OF
   TEXT RESULTS-TYPE *"
80 FOR T=1 TO 28:N(T)=Ø:
   NEXT T
90 INPUT "▪ENTER TEXT▪";A$
100 IF A$="*" THEN 180
110 PRINT "♥"
120 FOR I=1 TO LEN(A$)
130 FOR J=1 TO 26
140 IF J=ASC(MID$(A$,I,1))–64 THEN
   N(J)=N(J)+1
150 NEXT J
160 NEXT I
170 GOTO 90
180 PRINT "♥"
190 PRINT "▪LETTER▪FREQ▪▪L
   ▪▪▪F▪"
200 FOR I=1 TO 13
210 PRINT CHR$(64+I);TAB(6);N(I);
   TAB(13);CHR$(77+I);TAB(16);
   N(13+I)
220 NEXT I
```

```
20 MODE1:VDU 19,Ø,3,Ø,Ø,Ø,19,7,4,Ø,Ø,Ø
```



- CODE BREAKING
- LETTER DISTRIBUTION PROGRAM
- NEW CODING METHODS
- THE RAIL FENCE CODE
- MULTIPLICATION CODE PROGRAM
- USING MULTIPLICATION KEYS
- CODING DICTIONARIES
- SETTING UP A CODEBOOK
- PROGRAMMING YOUR OWN CODEBOOK SYSTEM

```
30 PRINTTAB(13)“FREQUENCY COUNT”’
40 PRINTTAB(16)“WARNING”’’
50 PRINTTAB(2)“DO NOT LEAVE SPACES
   BETWEEN WORDS”’’
60 DIM N(28)
70 PRINT“TO END INPUT OF TEXT AND
   OUPUT RESULTS—TYPE *”
80 FOR T=1 TO 28:N(T)=0:NEXT T
90 INPUT“ENTER TEXT”;A$
100 IF A$=“*” THEN GOTO 180
110 CLS
120 FOR I=1 TO LEN(A$)
130 FOR J=1 TO 26
140 IF J=ASC(MID$(A$,I,1))—64 THEN
   N(J)=N(J)+1
150 NEXT J
160 NEXT I
170 GOTO 90
180 CLS
190 PRINT“LETTER” TAB(8)“FREQUENCY”
   TAB(20)“LETTER” TAB(28) “FREQUENCY”
200 FOR I=1 TO 13
210 PRINTTAB(4) CHR$(64+I);TAB(12)N(I);
   TAB(24);CHR$(77+I)TAB(33)N(13+I)
220 NEXT I
230 END
```

```
20 CLS
30 PRINT@8,“FREQUENCY COUNT”
40 PRINT@76,“WARNING”
50 PRINT@134,“do not leave spaces”:
   PRINT@169,“between words”
60 DIM N(28)
70 PRINT@224,“TO END INPUT OF TEXT
   AND OUTPUT□□□□□□□□□
   RESULTS — TYPE *”
80 FOR T=1 TO 28:N(T)=0:
   NEXT
90 INPUT“ENTER TEXT□”;A$
100 IF A$=“*” THEN 180
110 CLS
120 FOR I=1 TO LEN(A$)
130 FOR J=1 TO 26
140 IF J=ASC(MID$(A$,I,1))—64 THEN
```



```
    N(J)=N(J)+1
150 NEXT J
160 NEXT I
170 GOTO 90
180 CLS
190 PRINT"LETTER□□□FREQ'
    □□□LETTER□□□FREQ'"
200 FOR I=1 TO 13
210 PRINTTAB(2);CHR$(64+I);
    TAB(10);N(I);TAB(19);CHR$
    (77+I);TAB(27);N(13+I)
220 NEXT
230 END
```

The operational structure of this program simply provides a counting mechanism. In the first phase of the program, 28 index variables are set equal to zero. These are used to store the frequency counts of the 26 letters. The final two variables are included in case you want to amend the program in order to produce percentages or other summary statistics.

## MULTIPLICATION CODES

During the American Civil War the Rail Fence code was used to send secret messages. It works like this: suppose you wish to pass on the commercially sensitive warning: SELL CONSULS SOONEST. Then, take a sheet of lined paper and write the first letter on the top line, the second on the second line, third letter on the first line, and so on to get:

S L C N U S O N S
E L O S L S O E T

—a pattern like the Western railroad fences. This message can then be encoded as SLCNUSONS ELOSLSOET or divided into more realistic word lengths as SLCNUS ONSELO SLSOET.

Now, the fence post code is really a special case of the more modern multiplication code. Ignoring spaces, the original plain text message contained 18 characters. These can be stored in a 2 × 9, 9 × 2, 3 × 6 or 6 × 3 array as shown.

Anyone trying to break the code without knowledge of the multiplication key would have a hard time. All that is really happening is that the message is written down the page until the first column of our array is filled, and then continued at the top of the second column. When the whole array is used up the encrypted text is read off across the page. Sometimes it is a good idea to add a few extra dummy letters to the end of a message—just to fool would-be code breakers.

The multiplication program again uses the MID$ function to provide a coding or decoding facility in just a few short lines (Lines 140–210). The multiplication code simply reads a message into an array in one direction and prints it out in the other; easy but effective.

```
20 BORDER 0: PAPER 0: INK 7: CLS
30 PRINT TAB (6);"MULTIPLICATION CODE"
40 PRINT : PRINT : PRINT
50 PRINT FLASH 1; PAPER 2;"DON'T LEAVE
   SPACES BETWEEN WORDS"
60 INPUT "ENTER TEXT"'m$
70 INPUT "ROWS ?□□";m
80 INPUT "COLUMNS ?□□□";n
90 INPUT "CODE (c) OR DECODE
   (d) ?□";e$
100 PAUSE 50: CLS
110 IF e$="c" THEN LET x=m
120 IF e$="d" THEN LET x=n
130 DIM d$(x,LEN m$/x)
```






**Using the multiplication code with several keys, you can encrypt the same message in different ways**

```
140 FOR i=1 TO x
155 LET a$="": LET m$=m$+"□"
160 FOR J=1 TO LEN m$-1 STEP x
180 LET a$=a$+m$(i+j-1
   TO i+j-1)
190 NEXT j
195 LET d$(i)=a$
197 LET m$=m$( TO LEN m$-1)
200 PRINT d$(i);: IF e$="c" THEN PRINT
   "□";
210 NEXT i
220 STOP
```

```
30 PRINT "■■>■■MULTIPLICATION
   CODE"
50 PRINT "■■■DON'T LEAVE SPACES
   BETWEEN WORDS"
60 INPUT "■TEXT■";M$
70 INPUT "ROWS■";M
80 INPUT "COLUMNS(C) OR
    DECODE(D)";INPUT E$
100 PRINT "■"
110 IF E$="C" THEN X=M
120 IF E$="D" THEN X=M
130 DIM D$(X)
140 FOR I=1 TO X
150 D$(I)=""
160 FOR J=1 TO LEN(M$) STEP X
170 B$=MID$(M$,I+J-1,1)
180 D$(I)=D$(I)+B$
190 NEXT J
200 PRINT D$(I);
210 NEXT I
```

```
20 MODE1:VDU 19,0,4,0,0,0,19,7,
   3,0,0,0
30 PRINT TAB(10)"MULTIPLICATION CODE"
40 PRINT'''
50 PRINTTAB(2)"DO NOT LEAVE SPACES
   BETWEEN WORDS"''
60 INPUT"TEXT";M$
70 INPUT"ROWS";M
80 N=INT(LEN(M$)/M)+1
90 INPUT"CODE(C) OR DECODE(D)";E$
100 TIME=0:REPEAT UNTIL TIME>150
110 IF E$="C" THEN X=M
120 IF E$="D" THEN X=N
130 DIM D$(X)
140 FOR I=1 TO X
150 D$(I)=" "
160 FOR J=1 TO LEN(M$) STEP X
170 B$=MID$(M$,I+J-1,1)
180 D$(I)=D$(I)+B$
190 NEXT J
200 PRINTD$(I);
210 NEXT I
220 PRINT''''':INPUT"AGAIN";
   AN$:IF AN$="Y" THEN RUN
230 END
```

```
20 CLS
30 PRINT@6,"MULTIPLICATION CODE"
40 PRINT:PRINT:PRINT
50 PRINT"DON'T LEAVE SPACES BETWEEN
   WORDS"
60 PRINT:INPUT"TEXT□";M$
70 INPUT"ROWS□";M
80 INPUT"COLUMNS□";N
90 INPUT"CODE(C) OR DECODE(D)□";E$
100 FORL=1TO1000:NEXT
110 IF E$="C" THEN X=M
120 IF E$="D" THEN X=N
130 DIM D$(X)
140 FOR I=1 TO X
150 D$(I)="□"
160 FOR J=1 TO LEN(M$) STEP X
170 B$=MID$(M$,I+J-1,1)
180 D$(I)=D$(I)+B$
190 NEXT J
200 PRINTD$(I)
210 NEXT I
220 END
```

## CODEBOOK

So far we have only considered ciphers. Proper codes—that is whole words or phrases which are encrypted by other words or numbers—are traditionally favoured by large organisations that operate from fixed premises. Embassies, ships and business houses fall into this category. Operating from fixed locations is preferable because one or more bulky coding dictionaries are necessary for translating the plain text.

The codebook program sets up a small sample code dictionary of 20 words. Using a two-dimensional array A$(I,J), in which I=1 indicates plain text and I=2 is coded text, the program first READS in the data displayed in the table. The next section (Lines 120–170 on the Commodores, Dragon and Spectrum. Lines 140–190 on the Acorn) takes in a word from the message and then prints out the corresponding entry in the table. If, for example, A$(1,6)=GO TO is entered, then A$(2,6)="10327" will be printed.

The larger message: DEPART FROM PARIS AT MIDNIGHT ON SATURDAY ARRIVE AT ROME AT DAYBREAK ON SUNDAY is coded as: 68677 90075 12128 26569 69783 27921 68719 12128 23874 12128 70355 69783 48553. The encrypted text: 74891 22317 12128 26569, translates to SEND MONEY AT MIDNIGHT.

With the simple program given it would be just as quick to perform the coding or decoding operation by hand directly from the table. However, once the number of words gets into the hundreds or even thousands, the time saving produced by the computer is enormous.

```
20 BORDER 0: PAPER 0: INK 7: CLS
25 POKE 23658,8
30 PRINT TAB (10);"CODEBOOK"
40 PRINT : PRINT : PRINT
50 DIM a$(2,20,10)
60 FOR i=1 TO 2
70 FOR j=1 TO 20
80 READ a$(i,j)
90 NEXT j: NEXT i
100 INPUT "DO YOU WISH TO CODE(0)
   □□□□□□□□□□□□□□□□□□
   OR DECODE(1)";x
```

```
11Ø CLS
12Ø INPUT "enter word ";m$
125 IF LEN m$>=1Ø THEN GOTO 13Ø
127 FOR n=1 TO 1Ø-LEN m$: LET
   m$=m$+"□": NEXT n
13Ø IF m$="*" THEN GOTO 28Ø
14Ø FOR t=1 TO 2Ø
15Ø IF m$=a$(1+x,t) THEN PRINT
   a$(2-x,t)
16Ø NEXT t
17Ø GOTO 12Ø
18Ø DATA "NEWYORK","LONDON",
   "PARIS","ROME"
19Ø DATA "ARRIVE","DEPARTFROM",
   "GO TO","ESCAPE TO","SATURDAY"
2ØØ DATA "SUNDAY","NOON",
   "DAYBREAK","MIDNIGHT"
21Ø DATA "NIGHTFALL", "IN", "AT",
   "ON","SEND"
22Ø DATA "MONEY","FOOD"
23Ø DATA "54982","73581","9ØØ75",
   "23874"
24Ø DATA "68719","68677","1Ø327",
   "4Ø476"
25Ø DATA "27921","48553","11Ø72",
   "7Ø355"
26Ø DATA "26569","74832","1Ø996",
   "12128"
27Ø DATA "69783", "74891", "22317",
   "98724"
28Ø STOP
```

```
3Ø PRINT "■■>■■CODEBOOK"
5Ø DIM A$(2,2Ø)
6Ø FOR 1=1 TO 2
7Ø FOR J=1 TO 2Ø
8Ø READ A$(I,J)
9Ø NEXT J,I
1ØØ PRINT "■■DO YOU WISH TO
   CODE(Ø) OR DECODE(1)":INPUT X
12Ø INPUT "■ENTER WORD■";M$
13Ø IF M$="*" THEN 28Ø
14Ø FOR T=1 TO 2Ø
15Ø IF M$=A$(1+X,T) THEN PRINT
   A$(2-X,T)
16Ø NEXT T
17Ø GOTO 12Ø
18Ø DATA NEWYORK,LONDON,PARIS,ROME
19Ø DATA ARRIVE,DEPART FROM,GO
   TO,ESCAPE TO,SATURDAY
2ØØ DATA SUNDAY,NOON,DAYBREAK,
   MIDNIGHT
21Ø DATA NIGHTFALL,IN,AT,ON,SEND
22Ø DATA MONEY,FOOD
23Ø DATA 54982,73581,9ØØ75,23874
24Ø DATA 68719,68677,1Ø327,4Ø476
25Ø DATA 27921,48553,11Ø72,7Ø355
26Ø DATA 26569,74832,1Ø996,12128
27Ø DATA 69783,74891,22317,98724
28Ø END
```







**In order to use the Codebook system, both the sender of the message and the recipient need to have a copy of a fixed dictionary**

```
2Ø MODE1:VDU 19,Ø,3,Ø,Ø,Ø,19,7,
   4,Ø,Ø,Ø
3Ø VDU 23,224,255,255,255,255,
   255,255,255,255
4Ø PRINT'TAB(16)"CODEBOOK"
5Ø PRINT'''
6Ø DIM A$(2,2Ø)
7Ø FOR I=1 TO 2
8Ø FOR J=1 TO 2Ø
9Ø READA$(I,J)
1ØØ NEXT:NEXT
11Ø INPUT"DO YOU WISH TO CODE(Ø) OR
   DECODE(1)";X
12Ø TIME=Ø:REPEAT UNTIL TIME>15Ø
13Ø CLS
14Ø INPUT"WORD";M$
15Ø IF M$="*" THEN GOTO 3ØØ
16Ø FOR T=1 TO 2Ø
17Ø IF M$=A$(1+X,T) THEN PRINT
   CHR$(224);A$(2-X,T)
18Ø NEXT T
19Ø GOTO 14Ø
2ØØ DATA NEWYORK,LONDON,PARIS,
   ROME
21Ø DATA ARRIVE,DEPART FROM,GO TO,
   ESCAPE TO,SATURDAY
22Ø DATA SUNDAY,NOON,DAYBREAK,
   MIDNIGHT
23Ø DATA NIGHTFALL,IN,AT,ON,SEND
24Ø DATA MONEY,FOOD
25Ø DATA 54982,73581,9ØØ75,23874
26Ø DATA 68719,68677,1Ø327,4Ø476
27Ø DATA 27921,48553,11Ø72,7Ø355
28Ø DATA 26569,74832,1Ø996,12128
29Ø DATA 69783,74891,22317,98724
3ØØ END
```

```
2Ø CLS
3Ø PRINT@1Ø,"CODEBOOK"
4Ø PRINT:PRINT:PRINT
5Ø DIM A$(2,2Ø)
6Ø FOR I=1 TO 2
7Ø FOR J=1 TO 2Ø
8Ø READ A$(I,J)
9Ø NEXTJ,I
1ØØ INPUT"DO YOU WISH TO CODE(Ø)
   □□□□□□□□□□□□□□
   OR DECODE(1) ";X
11Ø CLS
12Ø INPUT"ENTER WORD□";M$
13Ø IF M$="*" THEN END
14Ø FOR T=1 TO 2Ø
15Ø IF M$=A$(1+X,T) THEN PRINT
   A$(2-X,T)
16Ø NEXT
17Ø GOTO 12Ø
18Ø DATA NEWYORK,LONDON,PARIS, ROME
19Ø DATA ARRIVE,DEPART FROM,GO TO,
   ESCAPE TO,SATURDAY
2ØØ DATA SUNDAY,NOON,DAYBREAK,
   MIDNIGHT
21Ø DATA NIGHTFALL,IN,AT,ON,SEND
22Ø DATA MONEY,FOOD
23Ø DATA 54982,73581,9ØØ75,23874
24Ø DATA 68719,68677,1Ø327,4Ø476
25Ø DATA 27921,48553,11Ø72,7Ø355
26Ø DATA 26569,74832,1Ø996,12128
27Ø DATA 69783,74891,22317,98724
```

Although the program listed above is limited to 20 words, it can easily be amended to include more. If, for instance, you wanted to enter 50 words you will need to make the following amendments.

On the Spectrum, Commodore and Dragon/Tandy change 2Ø to 5Ø in Lines 5Ø, 7Ø and 14Ø and on the Acorn change 2Ø to 5Ø in Lines 6Ø, 8Ø and 16Ø. You will also need to enter the new codes and additional lines in the program.

In the existing Spectrum program you are limited to entering words of no more than 10 characters. If you wish to increase the maximum of characters to, say, 12, all you need to do is to alter 1Ø to 12 in Line 5Ø. This restriction is not included in any of the other programs.

# CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT.*

# COMING IN ISSUE 34...

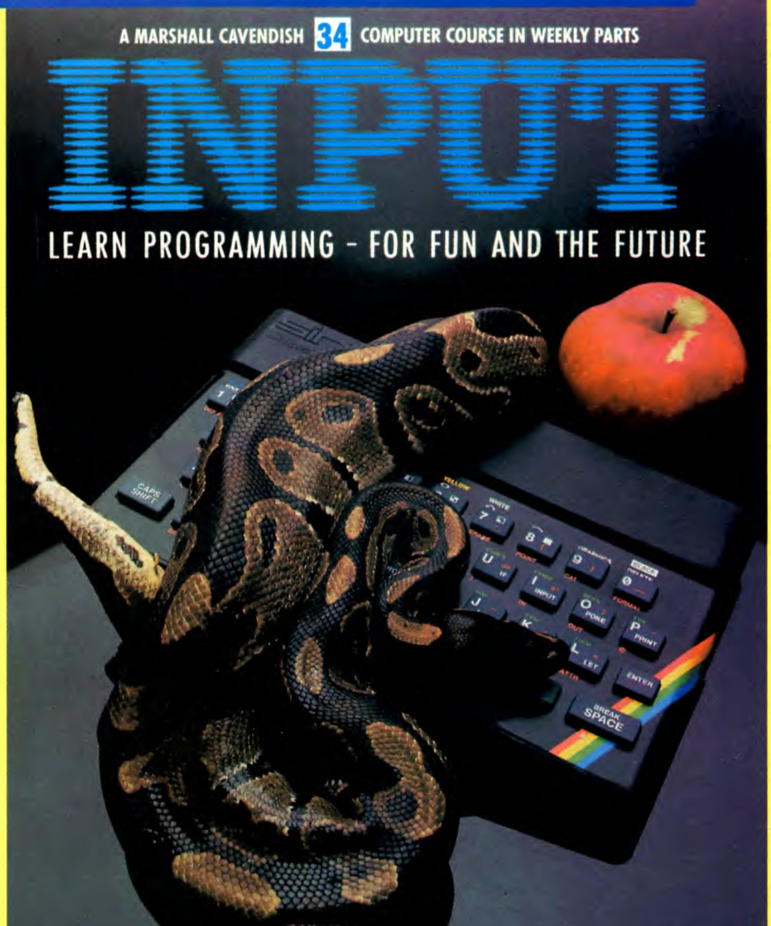
❑ *Computer modelling is a potent tool. See how mathematics can be used to understand GROWTH in nature*

❑ *Hit the jackpot with part two of FRUIT MACHINE, but don't look for piles of cash under your computer!*

❑ *It's both good and bad news for Willie in this part of CLIFFHANGER. Add rewards, potholes and snakes*

❑ *Plan Christmas 2084 with the completed DIARY and CALENDAR program ... or what about Bermuda in Summer 2105?*

❑ *BBC users can produce bright and colourful graphics, and save memory too, using TELETEXT screens*






**ASK YOUR NEWSAGENT FOR INPUT**